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

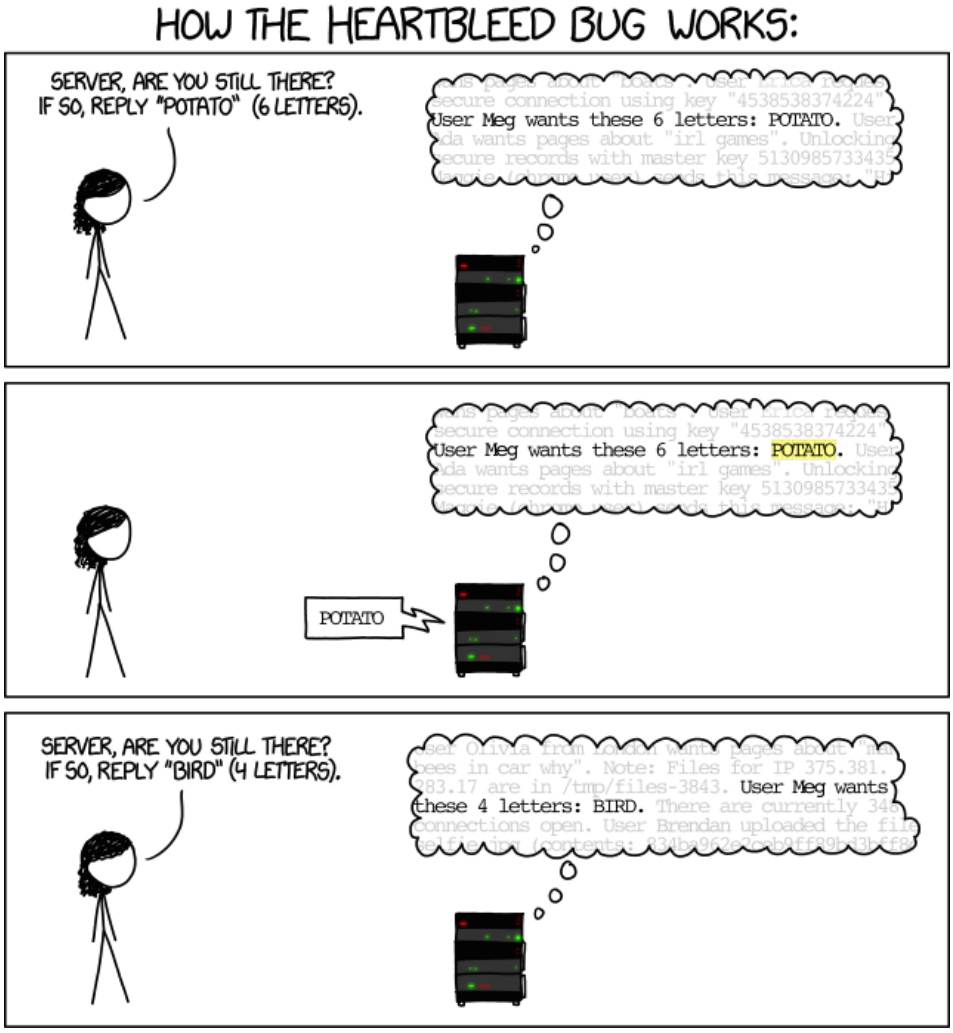
The Heartbleed Bug in OpenSSL (April 2014)

- 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.

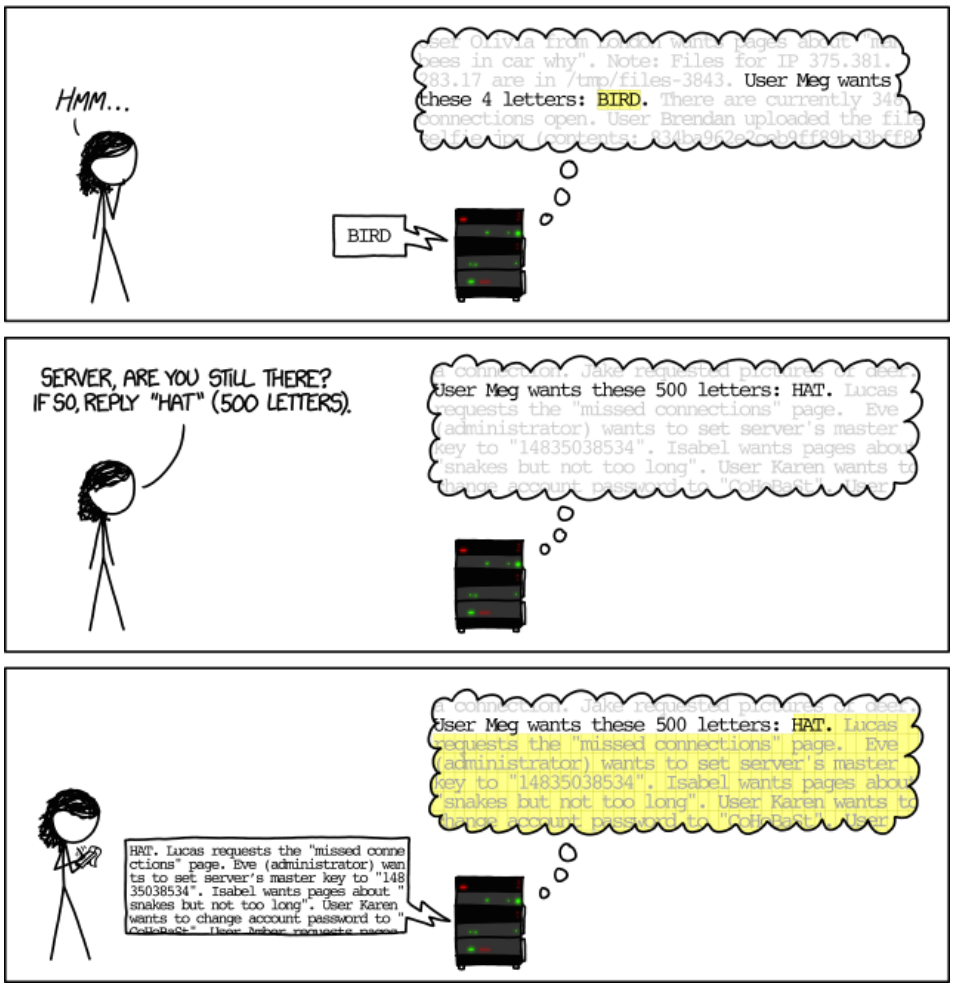
The Heartbleed Bug in OpenSSL (April 2014)

- 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



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 is to 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 Vulnerabilites

You can use this to read past what is accessible memory. So if your print format doesnt 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. `print("%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

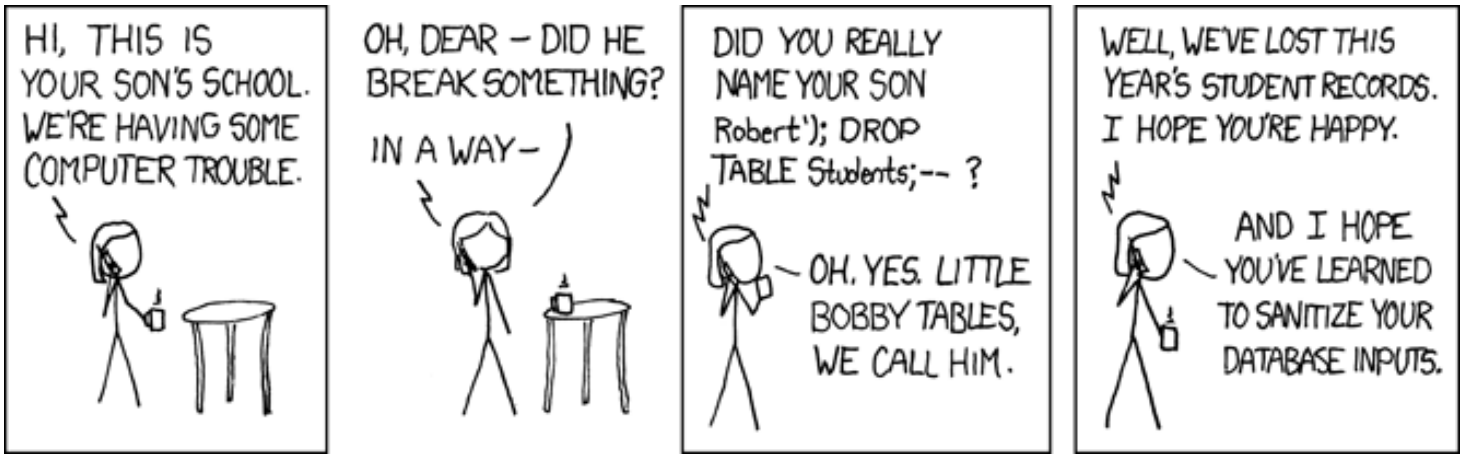
Incomplete mediation

- 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: `9876487423649248364924783648923649 2`
    - 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

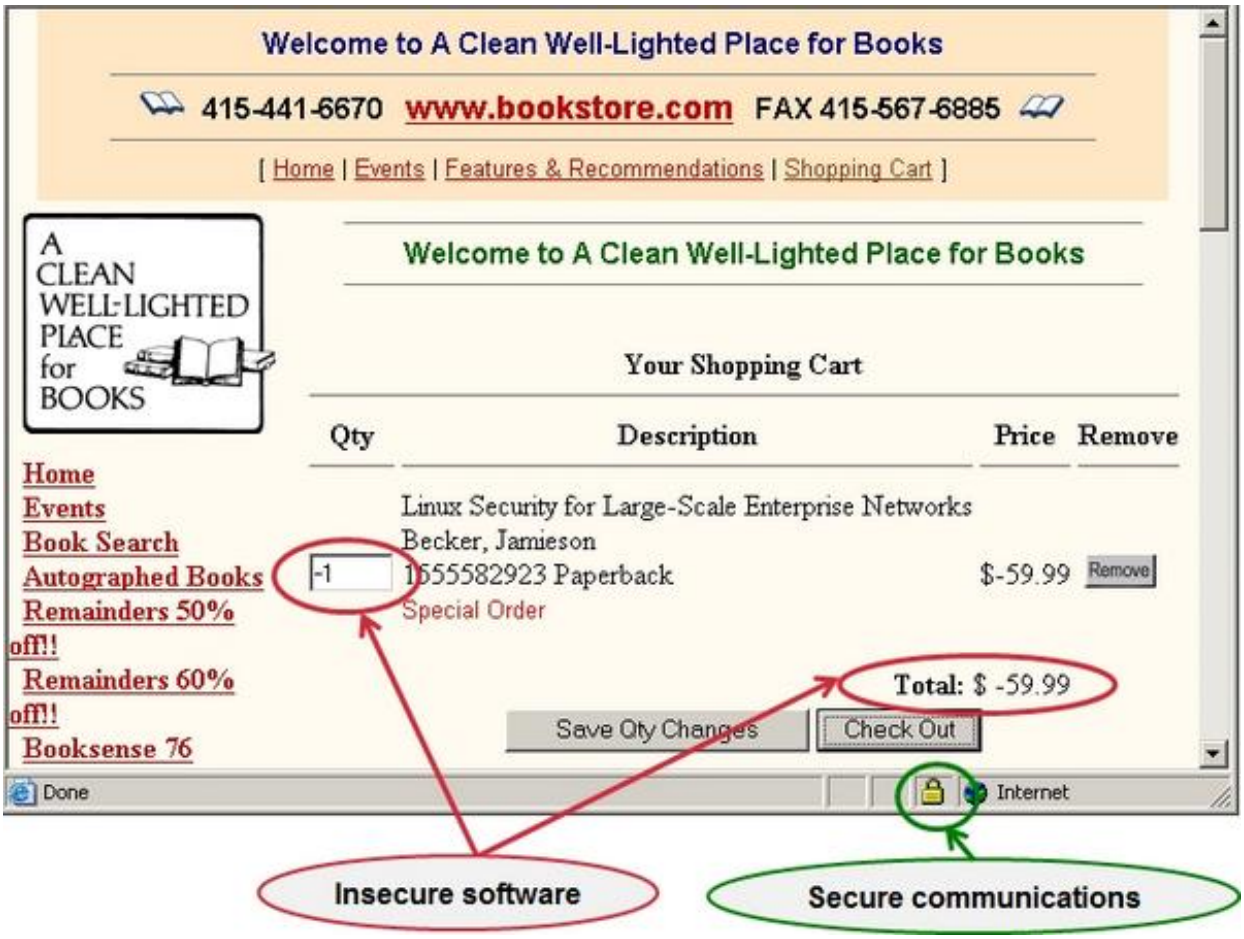
- The user can also modify any client-side state

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



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 reversable
- 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 see before. In behaviour based systems are prone to false positives, they can also have false negatives.

Base rate fallacy

- Suppose a breathalyzer reports false drunkness in 5% of cases, but never fails to detect true drunkness.
- 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

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

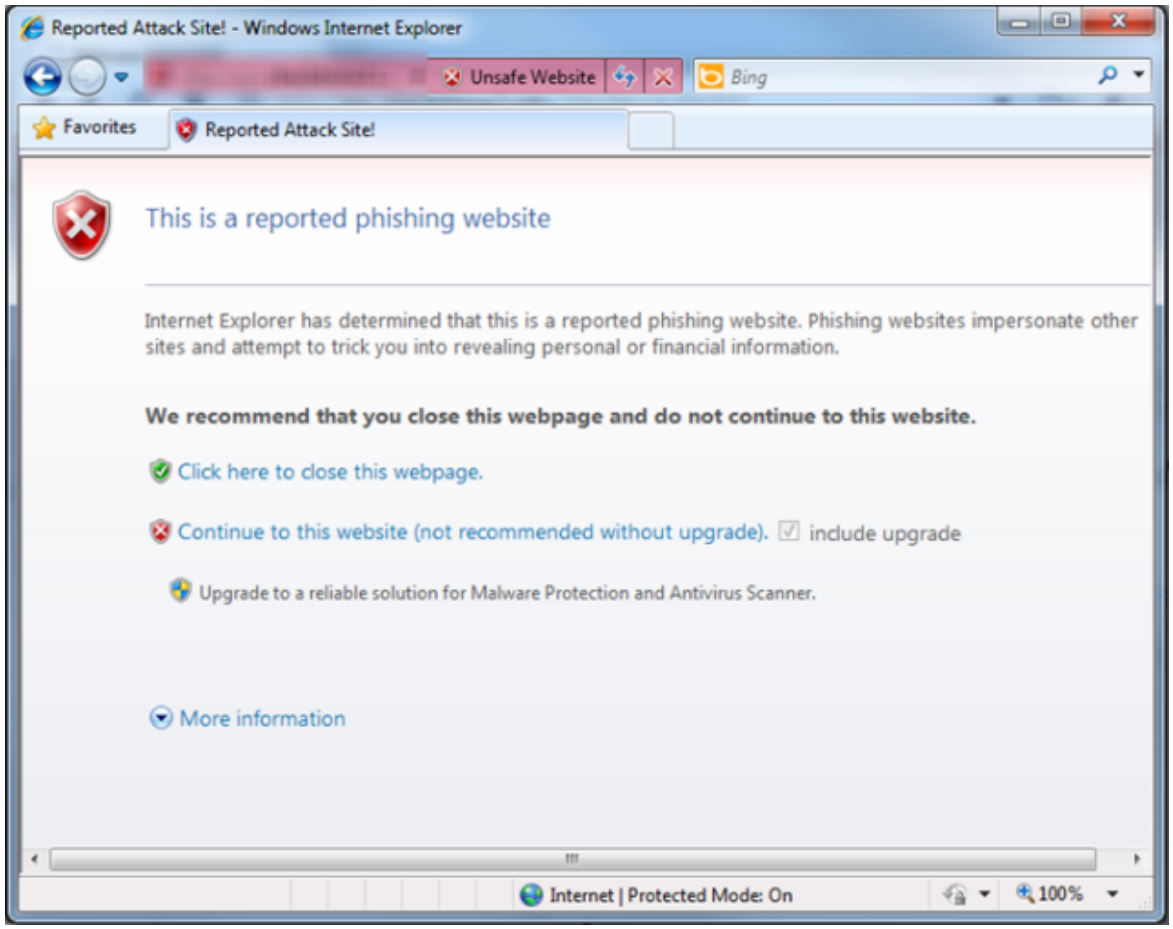
Trojan horses

- 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

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-ae

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 isnt 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.

	Virus	**Worm**
payload	yes	sometimes
distribution	requires user interaction	spreads by itself
location	attached to exe's	standalone
self-replicating	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

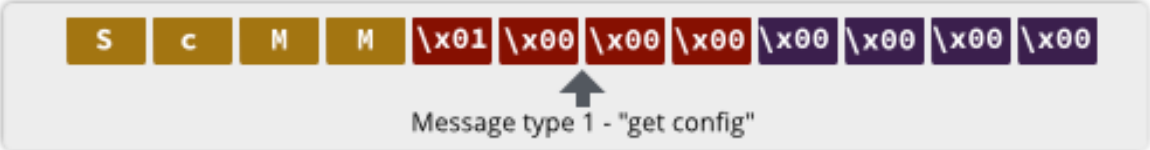
These are secret things that cause you browser to connect to a third party server. Usually a single pixel image or iframe. They can use this to get your ipaddress or store information in cookies. When you first go somewhere with a web bug it places a cookie on your browser so that when you go other places it can request that cookie and link where you've gone. Frequently this is used as adds to generate analytics about you. Addblock tends to just remove the adds 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

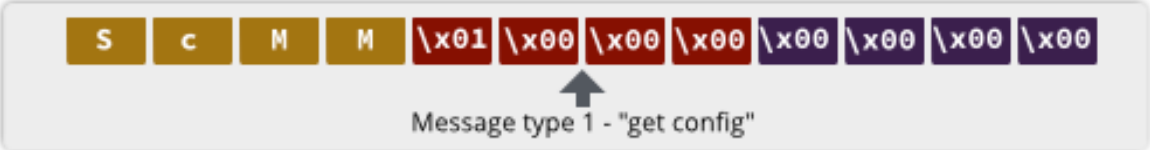
- Real examples:
  - Debugging back door left in sendmail
  - Back door planted by Code Red worm
  - Port knocking
    - The system listens for connection attempts to a certain pattern of (closed) ports. All those connection attempts will fail, but if the right pattern is there, the system will open, 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

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 a 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 baackdoor 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 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

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 / administator privileges on a machine (either starting with a local unprivileged account, or possibly remotely)
    - This method usually expoits 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 arent. 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 occured
  - 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 mod 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 stufff for that program. Its good to do with very sketchy data.