Paranoid Programming

Me: I've just been asked to do a class at TicketMaster.

Wife: How did you get TicketMaster to sell tickets for your class?

Paranoia

Just because your paranoid doesn't mean they aren't out to get you.

Paranoia

Murphy's Law:

If anything go wrong it will.

O'Shea's Law:

Murphy was an optimist

Paranoid Programming

Anticipating what can go wrong and devising ways to handle it

Paranoid Actions

- Think about everything that can go wrong.
- Avoid trouble if you can.
- Handle anything that comes your way.
- *Don't be surprised by anything that happens to you.*

Things to Make You Paranoid

- Bad input data
- Handling errors wrong
- Broken programs you must interact with
- Bad Computations
- Bad Security

Validate Input

- **If Bad Data In --> Bad Data Out**
- Programs which can never generate bad data – will.
- Your program is the one that has the bug until proven otherwise. (Especially if you're a junior programmer.)
- Prove the other guy wrong as early and as visibly as possible.

Validate Input

- Remember errors can be introduced any program that processes a transaction.
- Validation help you locate errors at the earliest possible time.
- ***Most importantly it makes sure your program works and you don't get blamed for someone else's error.***

Input Validation Aides

- Start files with magic strings

```
#Event Configuration File (V1.0)
```

- ➔ String tells you the file type
- ➔ Version information included

Other Magic Strings

- Magic Strings to start sections
- Magic Strings to end sections

Magic Record String

Record type

Unique Name

```
BEGIN seating chart (id=”the barn”)
    ROWS 32
    SEATS_PER_ROW 10
END seating chart (id=”the barn”)
```

XML Files

- XML is flexible
- XML is verifiable (if you write a definition file)
- XML is human readable (mostly)
- There are lots of libraries to parse XML files.

Magic Numbers and Data Structures

```
/* C Code */
struct record {
    int magic1;
    int data;
    // .. more data
    int magic2;
}

assert((r->magic1 == MAGIC1) &&
       (r->magic2 == MAGIC2))
```

Magic Numbers in Structures

- Someone can change the structure definition (and not tell you)
- Memory corruption will wipe out the magic numbers.
- Protects against packing problems.
- Protects against byte order problems.
- In C++ protected against a class being created with malloc.

Dealing with Validation Errors

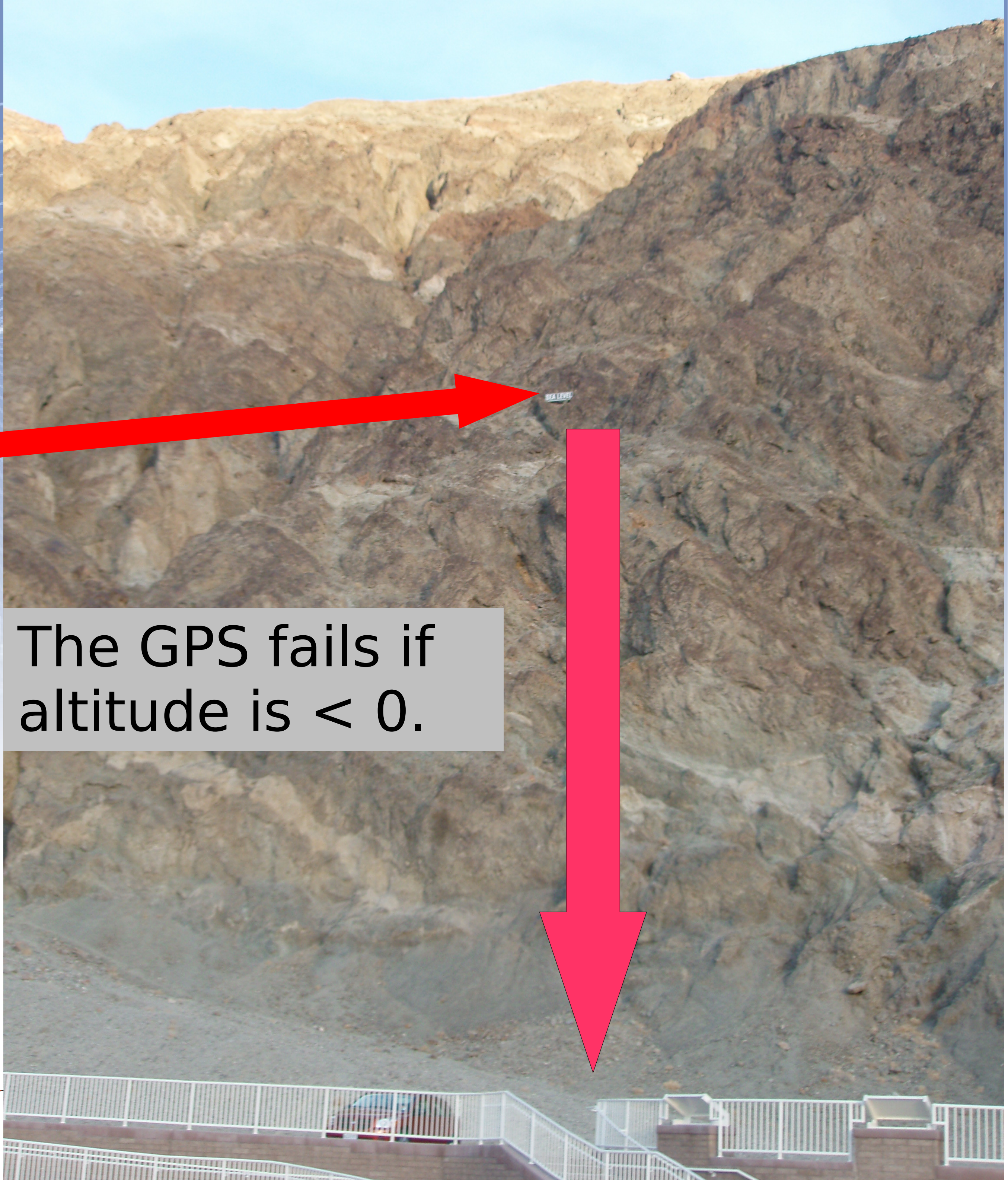
- Decide on the best course of action
  - Send error message to the user
  - Let the user retry
  - Take a standard fixup
  - Abort the operation
  - Log the failure
  - Raise a security alarm
  - Retry until the problem goes away

Validation Problems

I took my GPS to Death Valley and it died.

Badwater Death Valley

Sea Level



The GPS fails if altitude is < 0.

My GPS is here

Failure Mode

- Satellite status screen – Everything OK – I have a 3d fix.
- Map view – Everything is OK, but frozen.
  - (It's still 23 miles to the motel.)
- Coordinate page – Wrong location and altitude of 0.
- Reset – map view now shows that the GPS has no fix.
  - Satellite screen still shows 3d fix

Report an Error to the User

- Error messages should be clear.
  - Begin each error with the word: “Error”
- Keep things simple – Remember the person you are dealing with may have limited English.
- Give the user somewhere to go or something to do if possible.

Obscure by Design

- Error message:

```
Error: License file out of phase
```

- Designed to result in a technical support call, not avoid one.
- Much clearer (but useless version):

```
Error: Security violation.
Report yourself to customer service
immediately.
```

Take a standard fixup

- We're glad you want 10,000 seats for this concert.
- But the theater only hold 1,500.
- Besides there's a limit of 10.
- Order reduced to 10, do you want to continue?

Abort the Operation

- Before you abort
  - Log the error
  - Tell the user about the error

Something has gone horribly wrong with your order. Sorry but we have to abort and return to the main page.

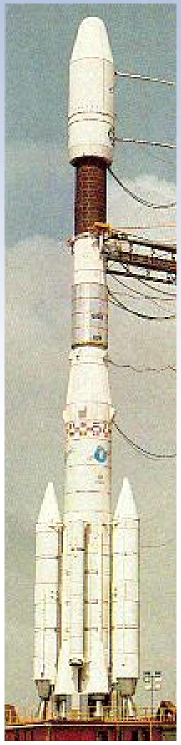
Sometimes Aborting is Bad

- The hardware platform had been upgraded.
- One process ran longer than normal.
- An overflow exception occurred.
- *There was no general exception handler.*
- The default exception handler halted the processor.

The Platform

Arian 4

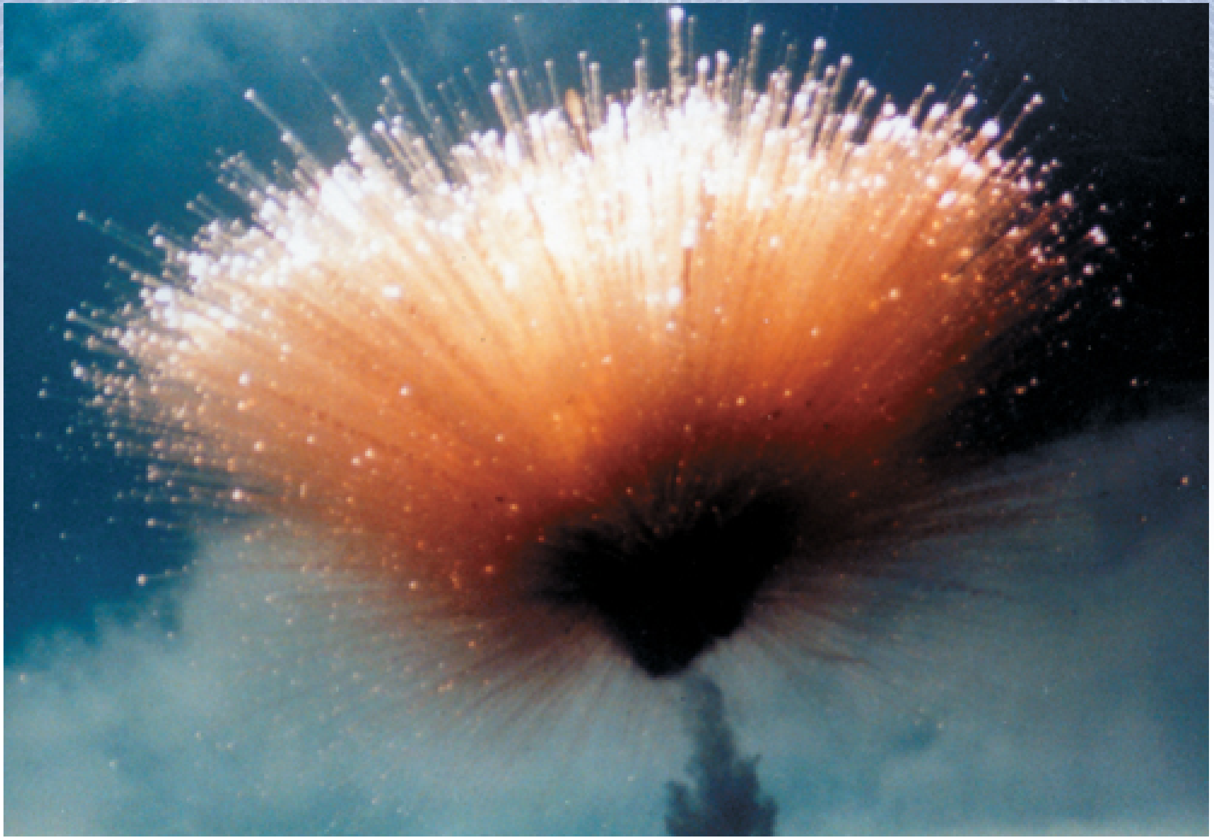
Arian 5



- One of the processor's jobs:
- Keep the rocket pointed into the air!

The System Did Not Crash

- The launch director blew it up before it could hit the ground!



All Failures goto the Logs

- Logs are especially useful in a web environment.
- Logs let you spot not only errors but also potential problems.
- You can do trend analysis on log data.
- More on logs later when we discuss security.

Raising a Security Alert

- I've have 20 orders in 5 minutes from the same user on the same IP address.
- No one can type that fast.
- Bot suspected.
- Note: Security alerts must be raised in a way in which they are acted upon. (More on this later.)

Alert Failure

- Phone company does a power failure drill and training sessions
  1. Power failure is simulated. Switch to city power turned off.
  2. Battery backup kicks in.
  3. Backup procedures are tested
  4. All mid and senior technicians go to class for a review of power failure procedures.
- *<u>Anyone see what step was left out?</u>*

Alert Failure

- After the drill someone forgot to switch on the city power.
- Batteries ran out of power.
- Technicians on site did not know what to do. (They weren't senior enough to require training.)
- Everyone who knew how to handle the problem was out of the office (at training).
- Result: No phone service for a major part of New York for almost a full day.

Playing well with others

- What do you do when a program you depend on fails?
  - Fail as well.
  - Keep trying until things work.
  - Try a backup system.

Keep Trying

```
while (1)  {
    $connect = DBI->connect($stuff);
    if (defined($connect)) {
       last;
    }
    log_problem("No db connection");
    sleep(DB_WAIT_TIME);
}
```

The Paranoid Database Runner

The Database Must Run!!!

1. Start database
2. If DB crashes, wait 30 seconds, restart
3. If DB hangs kill it nicely
    1. Wait up to 5 minutes. (Give DB time to stop.)
    2. Did DB die – Yes – Restart
    3. No – Kill harder (SIGKILL)
    4. Wait 30 seconds -- Restart

The DB Must Run

- Has the DB crashed more than 4 times in the last 30 minutes?
  1. Kill DB if needed
  2. Delete the entire database
  3. Start DB Program
  4. Recreate database from backup.

Transaction Failures

- You are in the middle of a transaction.
- The database fails.
- What do you do?
- If you are selling tickets:
  1. Possibly sell the same ticket twice
  2. Maybe leave a seat unsold

The answer obviously is:

- Sell the same seat twice
  - You get double the revenue
  - You can also sell tickets to the fight that's going to break out when the two ticket holders confront each other.

How do you know a transaction occurred

- I gave the DB a COMMIT command
  - The data could be in memory and not on disk.
- I read the data after I did the COMMIT.
  - You could have just read a in-memory copy.
- The DB says it wrote the data.
  - The OS could buffer the I/O in memory.
- After the COMMIT I did a fsync.
  - The disk has an on-board cache.

How do you know a transaction occurred

- I got out a hammer and chisel and carved it in stone.
  - An earthquake could cause the stone to shatter.

Does Delete mean Delete

- Does “rm *file*” get rid of the data.
  - No! It just puts the blocks in the free pool.
- Does “dd if=/dev/zero of=*file*” wipe out the data?
  - Maybe not. On some file systems data is written to a journal first, then the file.
  - Some file system have a versioning system built-in. You can go back in time.

Validate Use Input

- Turn on Perl's taint (-T) mode.
  - If you don't know what taint mode is, I'm going to get even more paranoid than I already am.

Check for permitted, not excluded

- Bad: Path can not contain “/../”

  Hackers: Use “/%2E%2E/” instead.

Check for permitted, not excluded

- Bad: The “number of seats” field must not contain letters or spaces.
  - I'd like -100 seats please at $29.95 a set. That's -2,995.00. So you owe me. Please send me a check.
- Good: The number is digits only.

```
if ($number !~ /^\d+$/) {
     # Handle invalid data
}
```

Validate Limits

- What's wrong with the following set of checks?

```
if ($number !~ /^\d+$/) {
    # Error
}
if ($number > LIMIT) {
    # Error
}
```

The Problem

- The following value is valid (at least in C and C++)

  4294967280

- In hex this number is:

  0xFFFFFFF0

- Converting the number to decimal:

  long int i = atol(4294967280);

- What does “i” contain?

  -16

You can't trust the machine to add 1 + 1.

True false test:

a) 1 + 1 = 2

b) 1/3 + 1/3 = 2/3

```
 0.3333
+0.3333
=0.6666
```

`This is not 2/3 (0.6667)`

You Can't Trust What You Can't See.

- Make out visible (Human Readable)
- You can't debug what you can't see

```
0000000 4b50 0403 0014 0000 0000 25f4 386c 2633
0000020 a8ac 002f 0000 002f 0000 0008 0000 696d
0000040 656d 7974 6570 7061 6c70 6369 7461 6f69
0000060 2f6e 6e76 2e64 616f 6973 2e73 706f 6e65
0000100 6f64 7563 656d 746e 702e 6572 6573 746e
```

Bad Memories

- What happens in perl when you do the following:

```
my @a; my $i;
$i = @a[32000];
```

- What happens when you do:

```
$a[3200000] = 5;
```

- How do you trap these types of errors?

Bad Memories

- What happens when you run out of memory?
  - You can trap out of memory errors in Perl, but you have to be clever about it.
  - Use `$^M` to allocate an emergency memory buffer.
  - Then trap `$SIG{__DIE__}`.

Running Out of Disk Space

- What do you do:
  - Abort.
  - Wait till space frees up.
  - Delete old data and hope space frees up.
  - Output the error message “Disk space 0K”.

PostgreSQL and Disk Space

- What happens when the DB runs out of disk space?
- PostgreSQL – You loose the DB.

```
if (free_disk_space(DB_DISK) < MIN_FREE) {
    stop_inserting_records();
    delete_old_db_records();
    start_inserting_records();
}
```

Disk Failures

- What happens if a disk goes bad?
- Do you have a backup for your critical data?
- How do you detect impending disk failure?
- Do you use the SMART capability of the disks?

Backup – What's Backup

- Major computer maker (at the time)
- Kept *NO* backups at all.
- I wrote a memo to the VP engineering:

  “A fire on the 15th floor will cause all our software to be destroyed. We need backup.”
- They installed fire extinguishers.

Disk Backup

- What is your backup policy?
- Is it written down?
- Has the backup system been tested?
  - Incomplete backups
  - Corrupt Backups

Dealing with Bad Disks

Procedure for dealing with bad disks

1. Remove disk from computer
2. Replace with good disk
3. If under warranty send to manufacturer
4. If past warranty pound it with a large hammer until it no longer resembles a disk
   - It will help avoid the temptation to reuse the disk
   - Plus it will make you feel better.

Hardware Backups

- System redundancy
- UPS and other emergency equipment.
- Don't buy cheap emergency equipment
  - Major cell phone maker
  - UPS systems bought used, very cheap.
  - 3 out of 4 unplanned outages for the year caused by UPS failure
  - 1 unplanned outage caused by the UPS catching on fire.

Paranoid Drill

- Where's the nearest fire exit?
- Where's the nearest fire extinguisher?
  - Bonus: Did you check the inspection tag on it?
  - Double bonus if you've ever reported an expired tag.
- The toilet is overflowing. Water is going down the hall toward the server room. Who do you call?
  - PS: Your computer's down. No looking things up on it.

Data Corruption

- Can you detect data corruption?
  - Magic Numbers
  - Checksums
- When is data backed up? How? By who? Where?
- RAID is not the answer to backups
  - A good RAID system means that you have multiple copies of the same bad data.

Security

- TicketMaster stores credit card information.
  - Loss of this information could result in massive identity theft
  - Loss of confidence in TicketMaster
  - Loss of future business
  - Loss of revenue
  - Recovery costs

Security

- Credit card information should be the second most secure information in the company
- The most secure information should be:

<u>The Log Files</u>

Log Files

- Tell you who tried to break in and how
- Tell you events leading up to a problem
- Allow you to identify trends and tune your system.
- Audit Trail

Protecting Security Logs

- Log information should go to a secure server.
  - All other services turned off (deaf and dumb)
  - Physically secure
- Who should see the log files?
  - Only a trusted few
- Who can modify the log files?
  - Only God with permission from two superiors.

Analyze the Information in the Logs

Respond Security Incidents

- Identify who is responsible for security
- Have a ***written*** procedure for dealing with security problems.

The EE Lab Incident

- EE Lab door is alarmed to prevent equipment from walking away.
- A student decide to see what would happen when the alarm was triggered.
- Security arrived in 2 minutes
  - Then didn't know what to do!
- So the students milling around told them.
  - and they followed their suggestions.

EE Lab Incident

- One student picked up a very expensive piece of test equipment and walked over to the confused security guards:

  "I think you should write down everybody's name who's here"

  He then walked away with his loot without giving his name. (He returned it an hour later.)

The Phone Maker Incident

- New directory of System Administration decides to show his people he knows what he is doing.
- He hires a security penetration test firm without telling anyone else.

The Incident

9:00 Penetration test team is let into the building by the director.

9:05 They unpack their equipment

9:30 Testing starts

9:31 The pagers of the duty sysadmin, the backup sysadmin, the security sysadmin, backup security sysadmin, and three other senior administrators go off.

The Incident

9:31-9:45 Sysadmins analyze the log files and determine that they have a security breach. They assemble in the “War Room” and pool their findings.

9:45 Security is called and told to assemble the strike team.

9:50 The conference room is raided by 20 security guards and three sysadmins.

Security Response

- Every step of the security response followed a written procedure.
- The procedure (and the call list) was on a small card everyone kept on the back of their badges.

Penetration Testing Nightmare

- Company hires security penetration testing firm.
- Company does not tell them that they are in a building that also houses MI5.
- MI5 picks up the team 3 blocks away.
- The second they walk in the door they are surrounded by heavily armed, very serious guards.

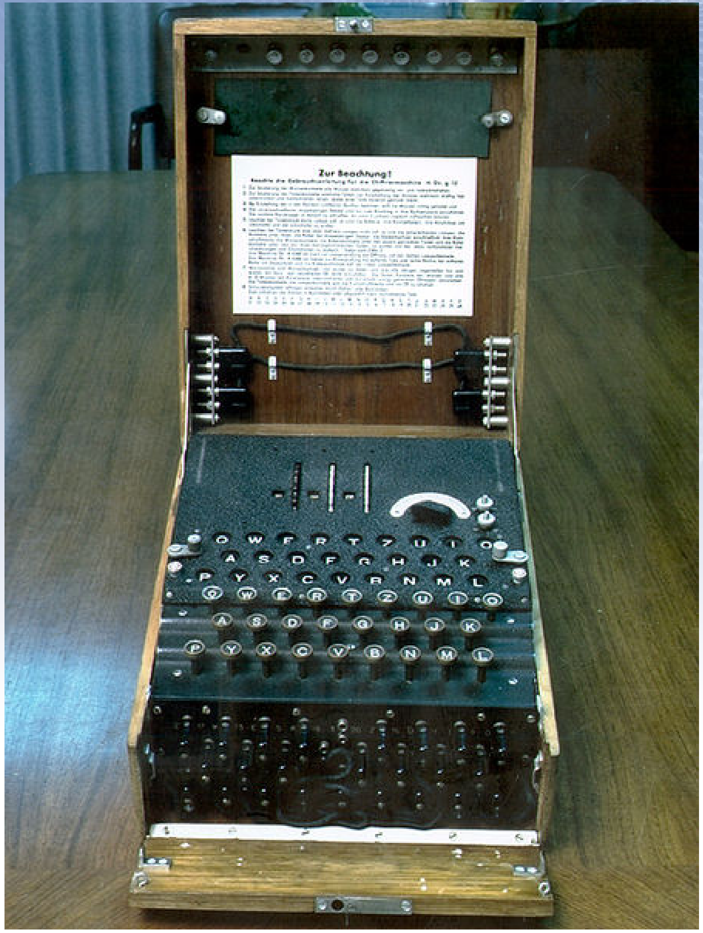
Security Procedures

1. Think about security problems before they happen.
2. Have multiple checks for security violations.
3. Make sure security violations are responded to.
4. Decide what the response is before the incident occurs.
5. Drill, review, and revise procedures.

Who will watch the watchers

- Who gets access privileges?
- How is this decided?
- How are access privileges revoked?
- Are key security people forced to take vacations?

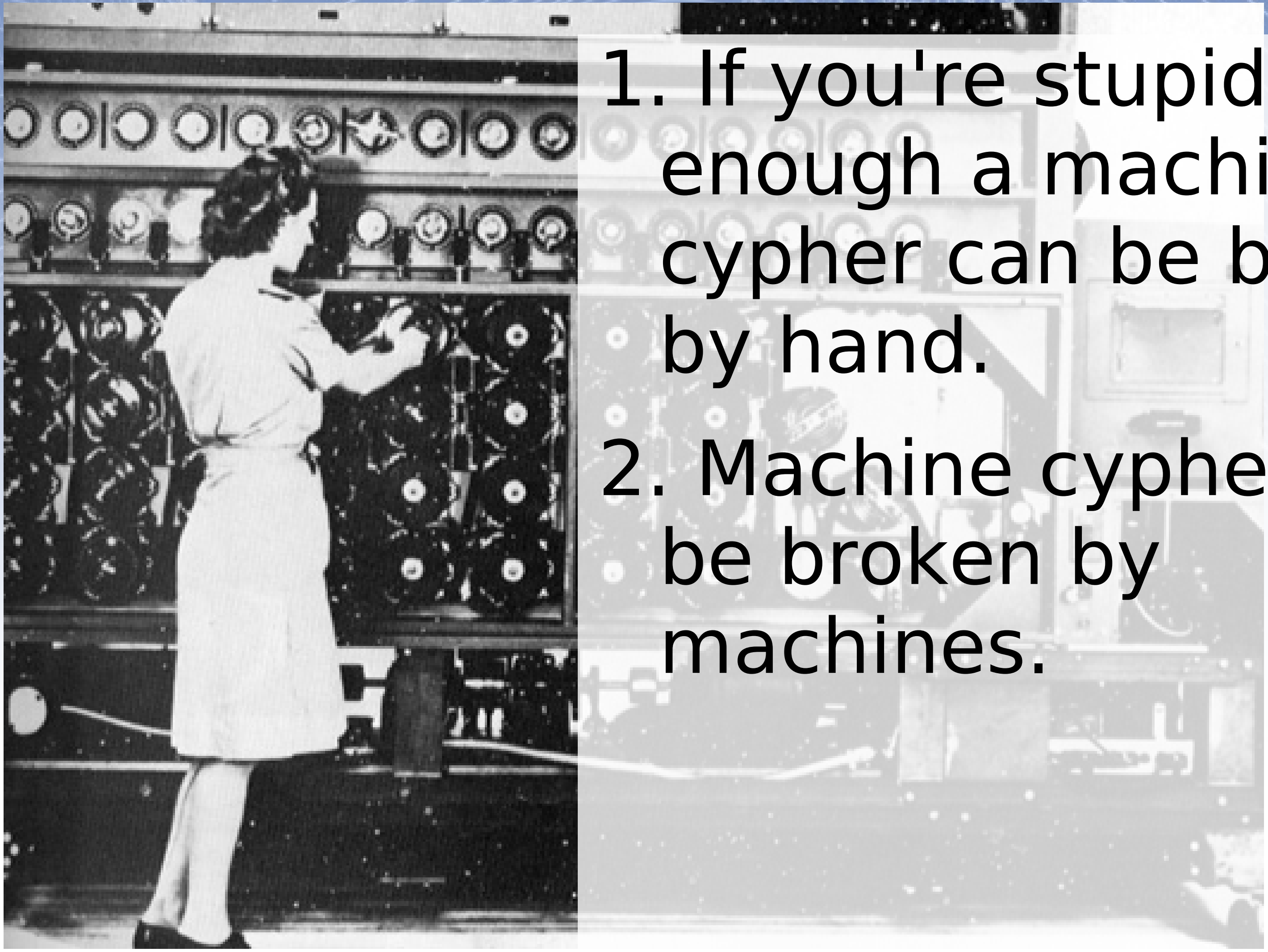
Secure Transmission -- Encryption



- German assumption:

A machine cypher can not be broken by hand.

The flaws in the assumption



1. If you're stupid enough a machine cypher can be broken by hand.
2. Machine cyphers can be broken by machines.

Secure Data Transmission

1. Assume hackers have access to your source code.
2. Use only well tested and well known encryption libraries. (Don't make up your own.)
3. Have your handling of sensitive data reviewed by someone who knows security.

Perl Paranoia

- Use all the built-in features programming features of Perl:

```
#!/usr/bin/perl -T
# (Taint)
use strict;
use warnings;
```

Prototypes

- Use prototypes

  1. It makes sure that you use the correct number of parameters.
  2. It performs very simple validation on the types of parameters you can pass in.

Perl and Error Checking

- Default (C style) Error handling
  - Functions return an error code (not all of them, but most of them)
  - What that code is is sometime wildly inconsistent.
  - Code must test each function call for error return and propagate it up the call chain.

Perl and Error Checking

- Problems with (C style) Error handling
  - Lots of code required to do error checking.
  - Programmers must deal with inconsistent error returns.
  - Problems occur when programmer get lazy and fail to error check everything.

Perl Exceptions

```
use Error;
try {
    do_something();
    if ($bad) {
        die(“Bad thing”);
    }
    if ($not_so_bad) {
        throw Error::Simple
            -text => “Not so bad”;
    }
}
```

Perl Exceptions

```
catch Error::Simple with {
    my $e = shift;
    print “Problem: $e->text\n”;
}
otherwise {
   # Problem, but don't halt the cpu
}
finally {
    clean_up();
};
```

Perl Exceptions

- Interface is very clean
- Less code is required.
- You can handle lots of different types of exceptions
- If you're going to use, use it everywhere.
  - Inconsistency will kill you
  - When I transitioned code from "error return" to exception, I named all exception throwing functions *{foo}*_e.

Consistency

- "In order to make things simple, this book has been split into three sections 1, 2, and C."

```
sub validate_customer_name($)
sub validate_country($)
sub validate_state($)
```

- What the name of the function to check the credit card number?

Key to Consistency

1. Do things the same way every time.
2. Write down the methodology you are using.

Unwritten rules mean that you have no rules!

3. Perform reviews to enforce the rules.

Small Example (Consistency)

```
sub foo($$$)
{
  my $name = shift; # Customer name
  my $card = shift; # Encrypted cc #
  my $exp = shift;  # Exp date (str)
```

The Rules

1. All subroutines must use prototypes.
2. All parameters are assigned names at the beginning of the function.
3. The shift method is used to name each parameter.
4. All parameter declarations must have a comment following them describing the parameter.

Perl and the Database

- School to parent: Did you really name your son:

```
John'; DROP TABLE STUDENTS;
```

- Always use prepared statements:

```
my $state = $dbh->prepare(
   “SELECT * FROM trans WHERE id = ?”);
my $result = $state->execute(@values);
```

Standards and Documentation

1. Standards must be written down.
2. The must be available to the programmers and *followed.*
3. You must be able to implement a standard.
4. Reviews must be conducted to make sure that people are in compliance with the standard.

Reviews

- Linus's Law: Given enough eyeballs, all bugs are shallow. -- Eric S. Raymond
- Reviews make for better, more secure (paranoid) programs.
- Reviews make for better, more secure (paranoid) *<u>programmers</u>*.

Checklists

- Checklists assure that procedures are consistent.
- They help make sure that all the steps are followed.
- Also by writing things down, other people can review them.

Reviewing the Checklist

You must review checklists to assure that:

- They are current.
- They are correct.
- They are relevant.

The Wrong Checklist

- Air Canada flight 143 had a broken gas gage. (Actually two out of three were broken but in a way that they couldn't tell how much gas was on board.)   FAILURE #1.
- Minimum flight requirements for a Boeing 767 state that must a have working gas gage.
- Pilot decided to use the alternate way of telling how much gas was aboard.
- FAILURE #2: Did not honor the first

Dipstick Method

According to the checklist provided by the airline the following procedure must be used:

1. Put a dipstick in the tank.
2. Convert gallons of fuel to pound of fuel.
3. Determine the number of additional pound of fuel needed.
4. Convert pounds of fuel to gallons of gas.
5. Get that much gas.

Double Checking

- Ground Crew double checked all the calculations using a second, independently created checklist.
- Both Pilot and Ground Crew Chief concluded that the plane had enough gas to reach the next stop:

22,300 Pounds

Wrong Checklist

- The 767 is an all metric aircraft

It needed 22,300 KG of fuel

- FAILURE#3: Using the wrong checklist.

The flight

- The bad news: Half way through the flight, the fuel pumps alarm: We're sucking air.
- The good news: The pilot flys glider planes as a hobby.
- The bad news: When the engine goes out the glass cockpit goes black.
- The good news: A few instruments stay up on batteries.

The flight

- The bad news: They can not make the nearest airport.
- The good news: The co-pilot trained at a nearby military airport, now abandon.
- The bad news: The “abandon” air strip was being used for drag racing.
- The good news: Race cars can get out of the way fast.

Results

- The plane landed safely.
- There was a minor fire in the nose wheel (which didn't lock).
- The racers had fire extinguishers to put it out.
- Some minor injuries caused by people jumping out the back of the plane.

AIR CANADA

Postscript

- Air Canada sent out a crew of mechanics to inspect the aircraft.
- They then send out a second crew to rescue the first one because:

They Ran Out of Gas

Learn from Your Mistakes

- When something fails find out why!
- Figure out how you got here.
- Is there a rule that would prevent this problem.
- Publish the results.
- Possibly adjust the standard.
- Make the same mistake once.

That way you only make new mistakes from then on.

Summary

- Assume that the everyone is out to get you.

  (It's better to be pleasantly surprised that rudely awakened.)

- Make sure you think of everything that can go wrong and plan as much of it as you can.
- Perform reviews so it's not just you that's paranoid.

Summary

- Have a written security plan for security problems.
- Assume that the hackers have a copy of your code.
- Have irregular security drills.
- Review security policies and procedures.

Programming

- Validate everything
- Test for all possible errors and most of the impossible ones.
- Use all the Perl tricks to make your code better:
  - use strict;
  - use warnings;
  - Turn on taint (-t)
  - Use prototypes.

Programming

- Use only prepared statements to access the database.
- Use the Perl “Error” module to improve exception handling.

Paranoia is a Group Effort

- Spread the paranoia
- Hold regular reviews of your code to make everyone as paranoid as you are.
- Review coding procedures as well as code.
- It's better to be afraid in a group than alone.

Questions?