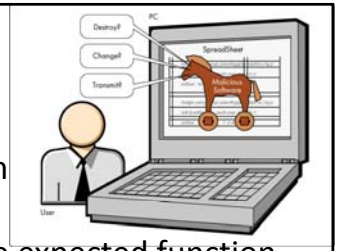
Malicious Logic

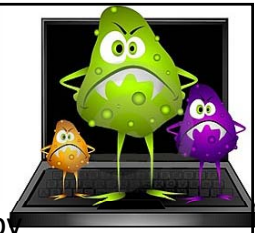
COMP620

Trojan Horses



- A Trojan horse is a program that does something malicious in addition to the expected function
- The author of the program intentionally adds code to do something in addition to what the user expects
- Often seen in “greeting card” programs

Computer Viruses



- A computer virus is malicious software that propagates itself by adding its machine language to other executables.
- A virus is very similar to a Trojan horse, except that the malicious code is added after the program is written.

Hiding Viruses

- Viruses can hide through encrypting themselves
- The initial code of the virus decrypts the instructions and then executes the main portion of the virus
- A polymorphic virus changes the code is a random like manner to avoid scanners

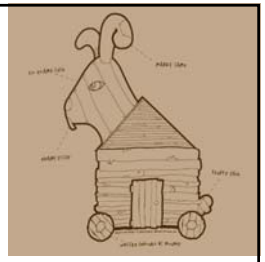
Macro Viruses

- Some file types support macros, such as Microsoft office
- Macros allow programmers to add functionality to the documents
- The functionality can be malicious

Computer Worms

- A computer worm is a program that propagates itself without modifying other programs
- Some worms are transported by email and will automatically send themselves to everyone on the victims address book

Rabbits



- Rabbits are programs that wastefully consume resources.
- These create denial of service attacks

Logic Bombs

- A logic bomb is like a Trojan horse. It is created intentionally by the programmer
- Usually a logic bomb waits for a particular situation and then does something malicious
- A classical logic bomb was written by a programmer in the payroll department. If his ID number did not appear when printing paychecks *(indicating he was fired)*, the program erased the payroll database

Anti-Virus

- Anti-virus programs provide two services
- Watch for suspicious activity, such as writing to an executable file
- Scans for **known** viruses
- Virus scanners are big string match programs
- The scanner reads files looking for machine language known to be in a virus.

Computer Theory

- Anti-virus and code checking software can **NOT** always determine if there is a virus or vulnerability

Halting Problem

- Can you write a program that inputs another program and some data to determine if the other program will eventually terminate (halt) when using the data?
- Proof by contradiction: Assume we have such a program (called Q)

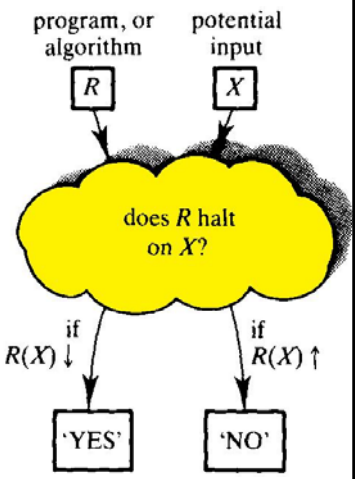


figure from "Algorithmics, The Spirit of Computing", 2nd ed. by David Harel

Halting Problem Proof

- Create program S calling program Q as a method.
- If Q answers Yes, go into an infinite loop

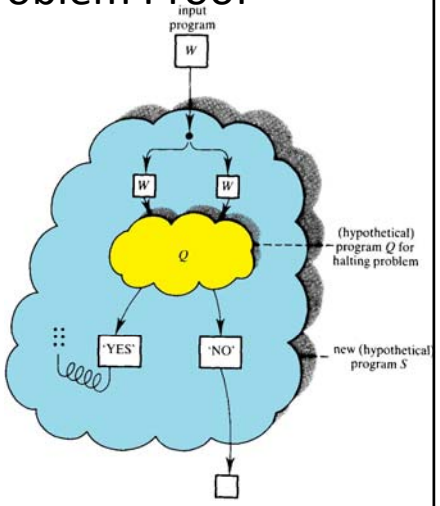


figure from "Algorithmics, The Spirit of Computing", 2nd ed. by David Harel

Halting Problem Proof

- Call program S using S as the input
- If Q says S halts, then it doesn't
- If Q says S does not halt, then it does
- This is a contradiction
- Therefore Q cannot exist

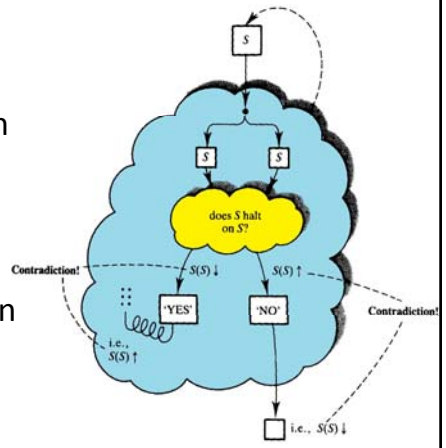


figure from "Algorithmics, The Spirit of Computing", 2nd ed. by David Harel

Halting Program

- Imagine you have a method that can tell if a program will halt with given data

```
boolean willHalt( program, data );
```

- This hypothetical method will return true if the input program would run to completion and false if the input program would run forever.

checkProg

- Consider this method that runs willHalt on the program with the program as data

```
boolean checkProg( program ) {
  if  ( willHalt(program, program)
  ) {
     while (true) { }  /* infinite
  loop */
  } else {
     return false;
  }
}
```

Contradiction

- Now imagine we call the checkProg method passing the method itself as input.

```
checkProg( checkProg );
```

- This creates a contradiction.
- Therefore the hypothetical willHalt function cannot exist.

Sometimes but not Always

- You can frequently solve a problem known to be non-computable
- The simple Hello World program will halt
- `while (true) {}` never halts
- While you may be able to solve specific instances of the problem, you cannot write a program to always solve the problem

Halt Function

- Imagine programs will halt when they execute the halt function
- We know that you cannot always tell if a program will halt, so you cannot always tell if a program executes the half function
- If you cannot tell if a program will execute the halt function, then you cannot always tell if a program executes a different function or any line of code

QED*

- If you cannot always tell if a program will execute a specific function or line of code, you cannot tell if the program will do something malicious

* the Latin phrase **quod erat demonstrandum**, which means *"that which was to be demonstrated"*