TK2100: Informasjonssikkerhet Lesson 04:Malware

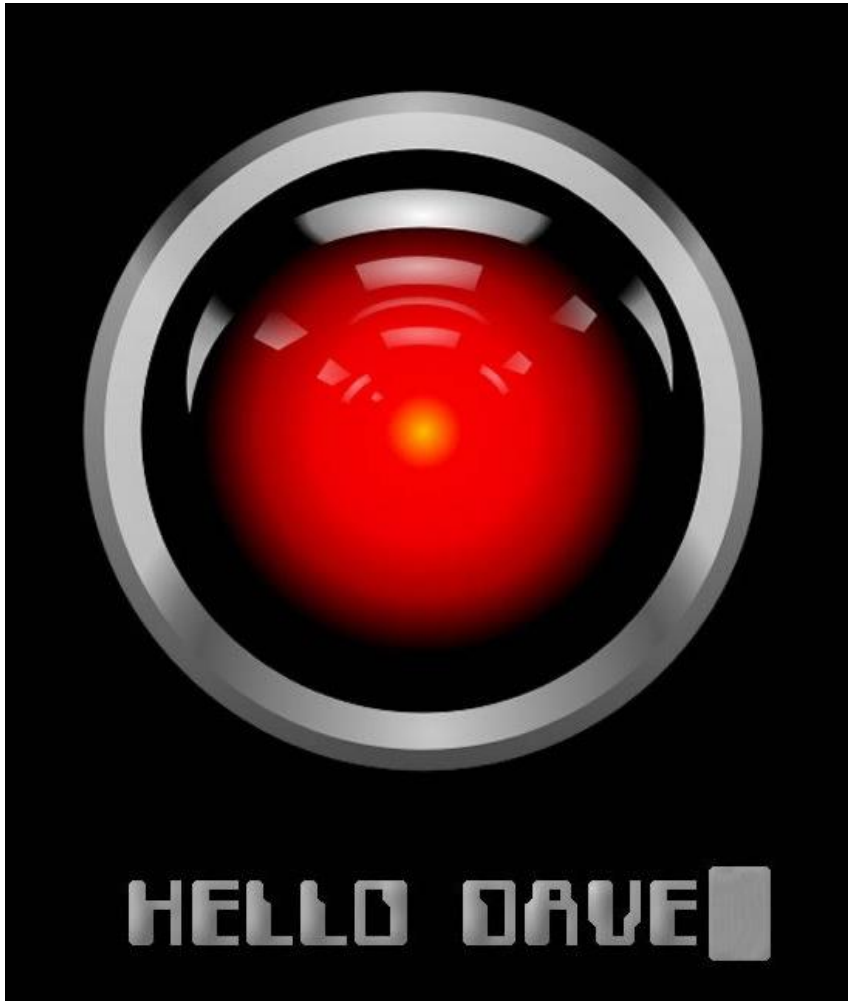
Dr. Andrea Arcuri
Westerdals Oslo ACT
University of Luxembourg

Goals

- Learn bases of how malware works and distribute
- Learn bases of how to protect from them
- Note: writing and analyzing malware is quite complex, as you need to have a good grasp of C language (which at this point in time you likely do not have) and low level details of OS… as such, this class regarding malware is mainly on the "*theoretical*" side…
- JavaScript: as we will need to use JS in the following lectures, we will do a intro/revision here as well

What Is Malware?

- Malware means **Mal**icious Soft**ware**
- Software that performs unauthorized and (most often) harmful actions
- If you haven't watched yet, highly recommended to see "*2001: a Space Odyssey*" and how the HAL computer behaves…



Malware Classification

- We can categorize malware into different types based on how it is spread and how it hides.
- Spreading
  - Virus: human-assisted spread (eg opening of email attachments and memory sticks)
  - Worm: Automatic spread from machine to machine over the net
- Hiding
  - Rootkit: Changes OS to hide presence
  - Trojans: Utility program that conceals malicious operations (eg keylogger)

How common is malware (2017)?

- 1 out of 131 emails contains attached malware
- 357 **M**illion variants
- Source: Symantec Internet Security Threat Report (ISTR) 2017

Insider Attacks

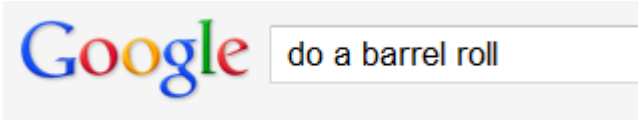
- “*In the case of malware, an insider attack refers to a security hole that is created in a software system by one of its programmers*”
- Different kinds of *Insider Attacks*:
  - Backdoors
  - Easter Eggs
  - Logic Bombs

Backdoors

- Hidden features, activated with special commands
  - eg, a way to bypass authentication, eg access online banking of different users
- Deliberately added by a developer
- *Debugging*: sometimes backdoors added during development to simplify testing/debugging
  - eg, skip authentication
  - should never end up in production configurations
- *Bug*: backdoors are not always obvious… often deliberate bugs that can be exploited as backdoors (eg buffer-overflow)… if malicious developer get caught, can just claim ignorance…

Easter Eggs

- Similar to backdoors, these are hidden features that can be activated with secret passwords or unusual sets of inputs
- Usually harmless, just done for “fun”
- Eg, try typing “*do a barrel roll*” on Google Search



Logic Bombs

- A program that performs a malicious action as a result of a certain logic condition
  - Eg, erase all data one month after an employee has been fired
  - Eg, disable authentication checks at certain time during the day
- *Extortion*: once logic bomb activated, provide solution via a backdoor after an extortion payment

Insider Attack Prevention

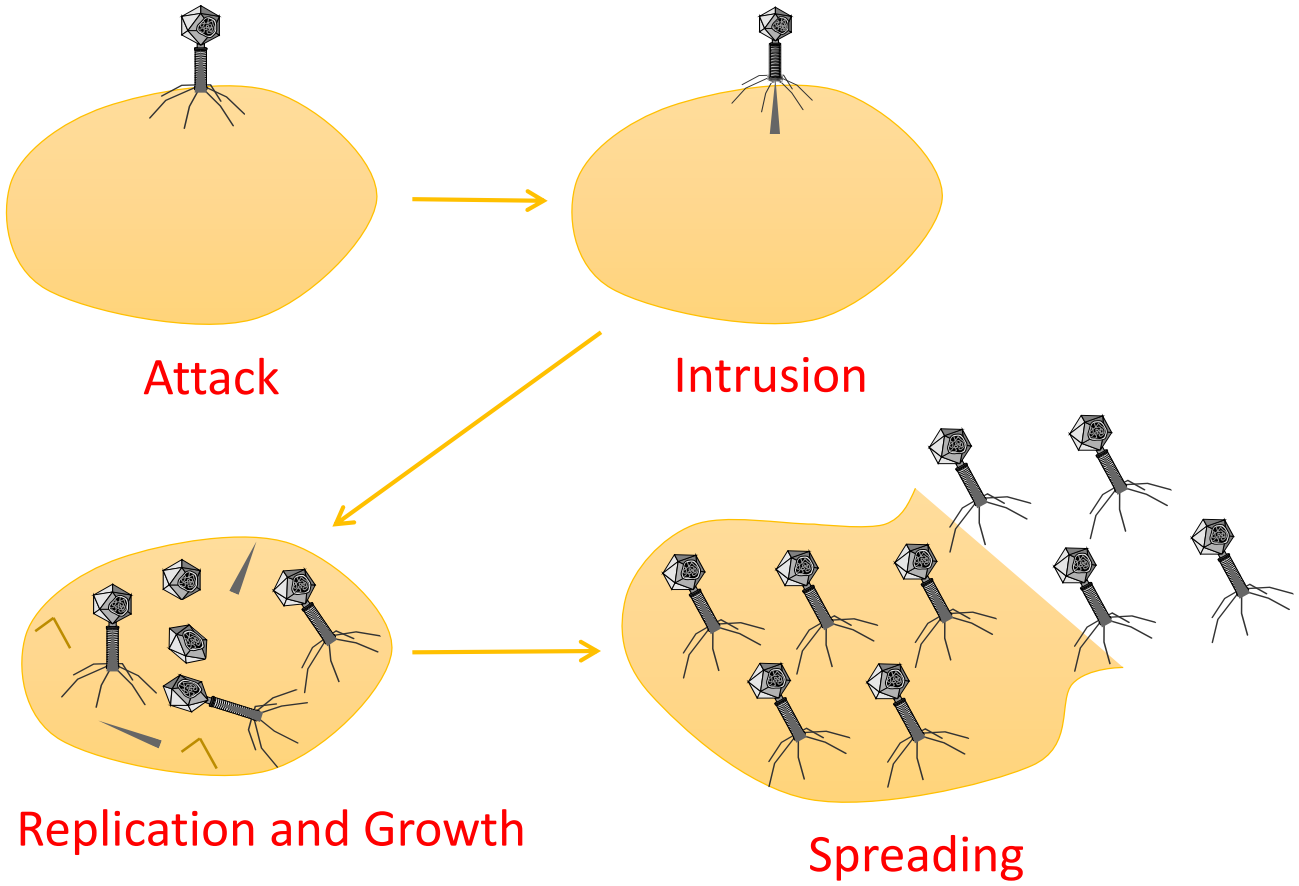
- No 100% solutions, but can reduce risk
- Avoid single points of failure, eg only one employee handling critical systems / backups
- Code-reviews: code written by a developer should be reviewed by a second one
- Static analyses: exist tools that can automatically check source code for some types of backdoors
- Etc.

Computer Viruses

What is a Computer Virus?

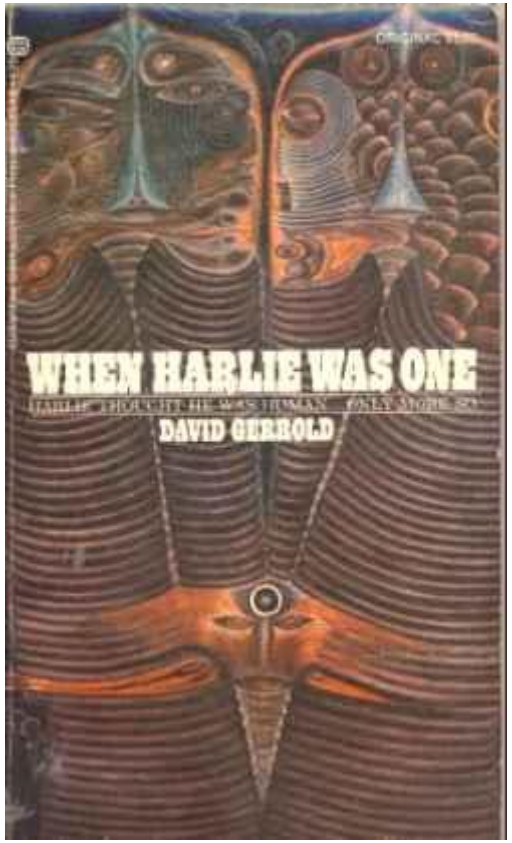
- A program that can replicate itself
  - by changing other files / programs
  - by infecting them with code
  - can modifying itself further
- The ability to infect existing files/programs is what separates viruses from other types of malware
- Generally requires initial user interaction
  - Click on a link and start installation
  - Open an email attachment
  - Share a memory stick, or other USB device

Similar to a Biological Virus



History

- In David Gerrold's AI novel "When HARLIE was One" (1971), the program VIRUS reproduces itself
- Adleman (The A in the RSA Encryption Algorithm) suggested the term (1984) for Fred Cohen, who wrote his PhD on the theory behind
- The first PC viruses observed in the 1980s were typically "boot-virus" that infected the boot sector on floppy disks and (eventually) hard disks.

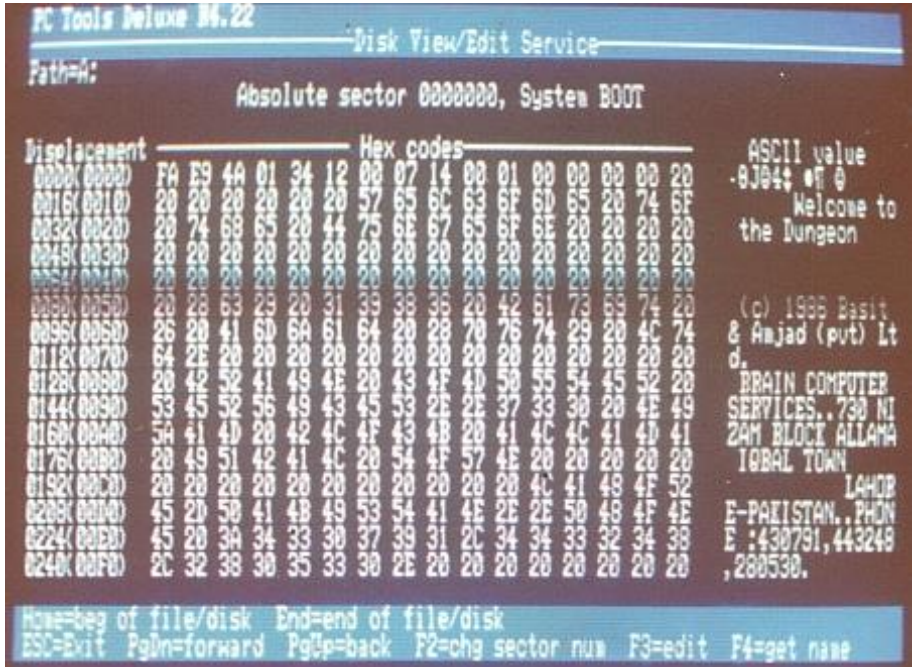


Traditional Viruses

- Today Worms and Trojans are more common
  - malicious files that are able to live an independent life without a host process
- Has existed longer than PCs, and first registered malware came in 1971 and is called "The Creeper Program" and spread over ArpaNet
- Brain is credited as the world's first "PC virus" (1986), and later that year, for the first time, it was possible to infect “.exe” files with Suriv-02
  - “.exe” files were initially considered a safe format because it was so complex that no one could manage to infect them, unlike other execution files that were using clear machine code ...
  - Some credit Old Yankee as the first “.exe” file infector
- Several dangerous viruses came out at this time:
  - AIDS Trojan (1989); encrypted your entire disk
  - Dark Avenger (1989); overwritten random parts of the disk 1/16 times the virus ran
  - Jerusalem (1987); Deleting files on machine on Friday 13th ...

Brain

- ©Brain – January 1986
- Written by two brothers in Pakistan to protect copyright on the program of a heart monitoring device they were selling
- Contained their phone number to remove the virus
- Spread via floppy disks when installing the program they wanted to protect



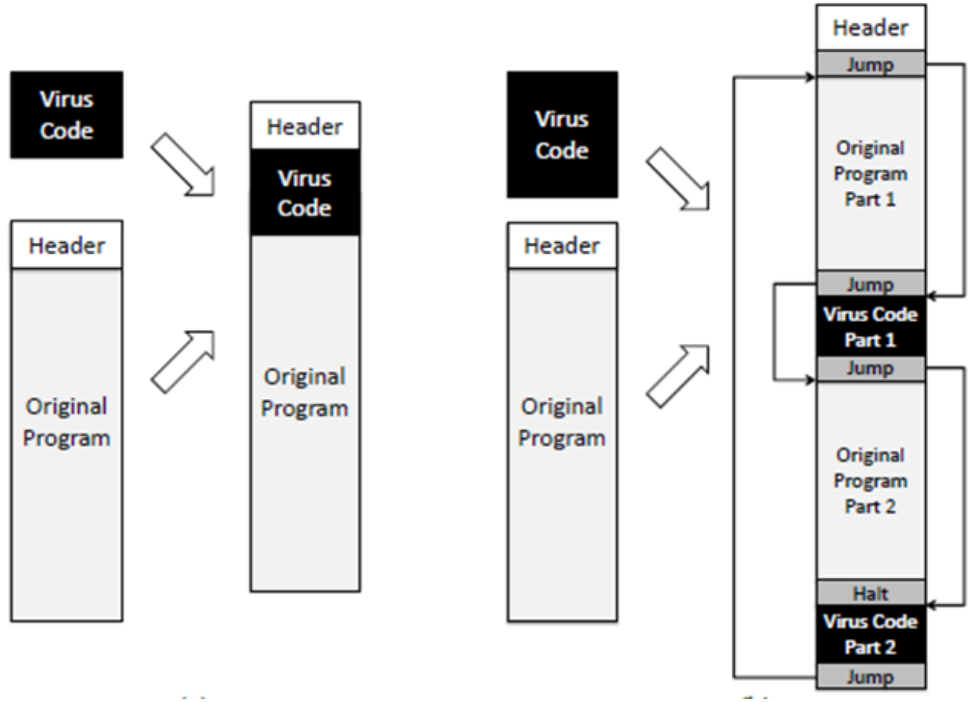

```
Welcome to the Dungeon © 1986 Basit * Amjad (pvt) Ltd. BRAIN
COMPUTER SERVICES 730 NIZAM BLOCK ALLAMA IQBAL TOWN LAHORE-
PAKISTAN PHONE: 430791,443248,280530. Beware of this VIRUS....
Contact us for vaccination...
```

Virus Phases

1. *Dormant*: virus laying down to avoid detection (eg, from an Antivirus software)
2. *Propagation*: replicates itself, infecting new files in new systems
3. *Triggering*: logical condition that causes switch from dormant to propagation phase
4. *Action*: do a malicious action, usually called *payload*
   - eg, delete/encrypt files

Types of Viruses

- Program Virus
  - Injecting itself inside the code of an existing program
- Macro Virus
  - Injecting documents like Words, which does support scripting code, usually called “macros”
- Boot Sector Virus
  - Infecting “boot sector”, which is what runs first when the OS starts



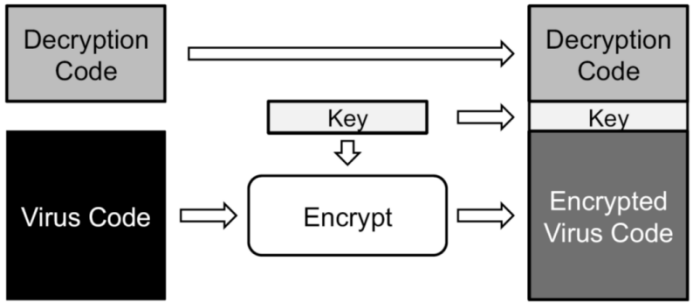
Virus Protection

- Simply, use an AntiVirus program… which needs to be updated often…
  - Why updates? Why often? How does it work?
- *Virus Signature*: once a virus is spread and known, can analyze its characteristics (eg special sequences of malicious instructions it uses), and then use *pattern-matching* to check if a file contains such instructions
  - As new viruses come up all the time, need to update the list of virus *signatures* in the antivirus program

Arms-Race

- Antiviruses use virus signatures to detect the viruses
- Virus developers try to find new ways to *hide* the presence of the viruses
- Antiviruses “evolve” to be able to handle these cases, and so on in an “arms-race”

Hiding Techniques



- Encrypt part of the code of the virus, and then decrypt and execute those parts at runtime when the infected program is running
  - A program that does decryption is hence “suspicious”
- *Polymorphic* virus: uses encryption, but each time it replicates, uses a different key
  - so signature of encrypted code is different
- *Metamorphic* virus: no encryption, but *code obfuscation* at each replication via code reordering and adding non-used instructions

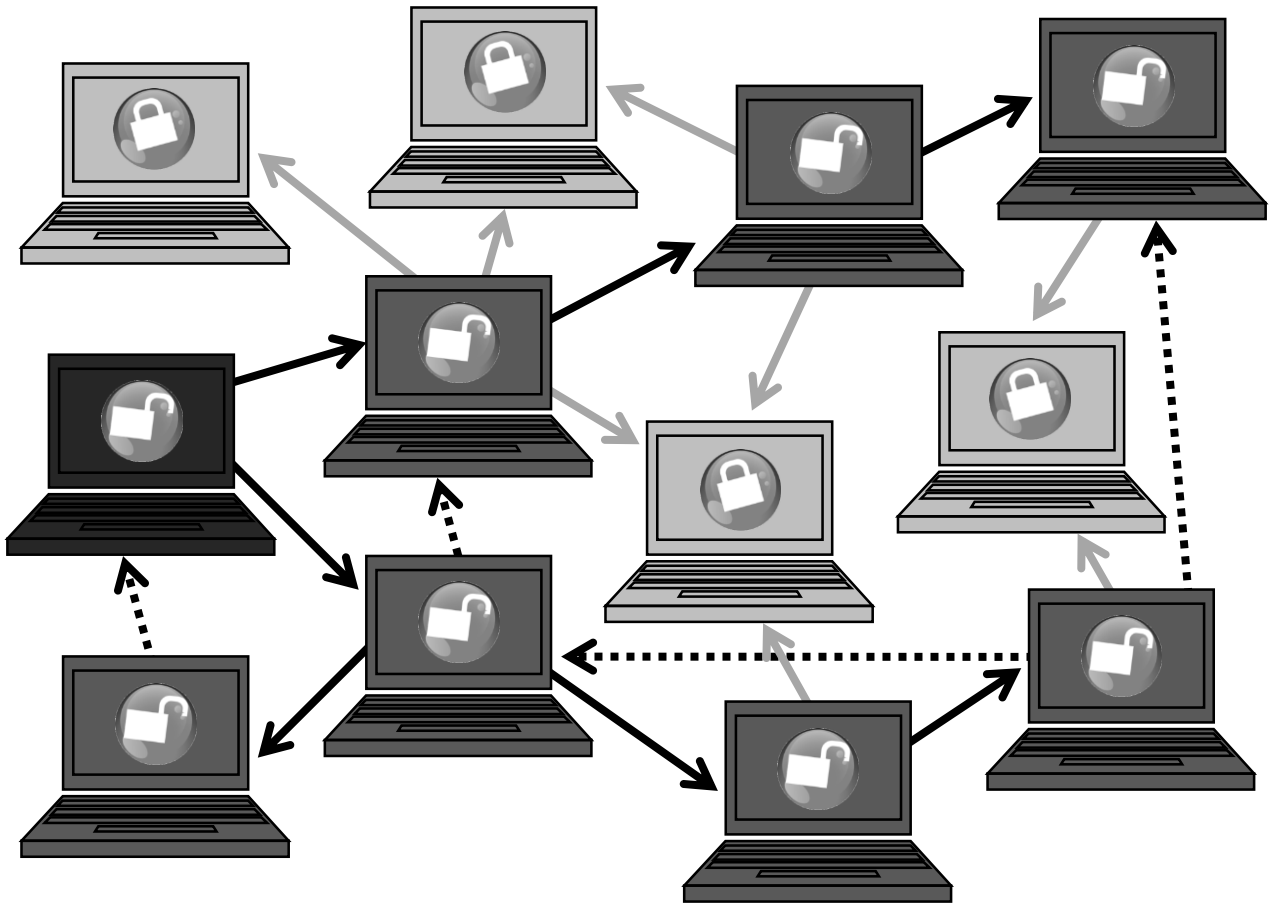
Worms

What is a Computer Worm?

- A malware that can replicate itself
- Does not need to inject its code inside other programs/files
  - Ie, this is contrast to viruses
- *Usually* does not need manual intervention (like opening an attachment in an email)
  - eg, exploiting security holes in existing programs, like a buffer overflow in web servers

Spreading of Worms

- Worms spread by finding and identifying vulnerable host machines
  - eg, having known security hole
- It must determine if the machine is vulnerable
- It must be able to check if the machine is already infected



History

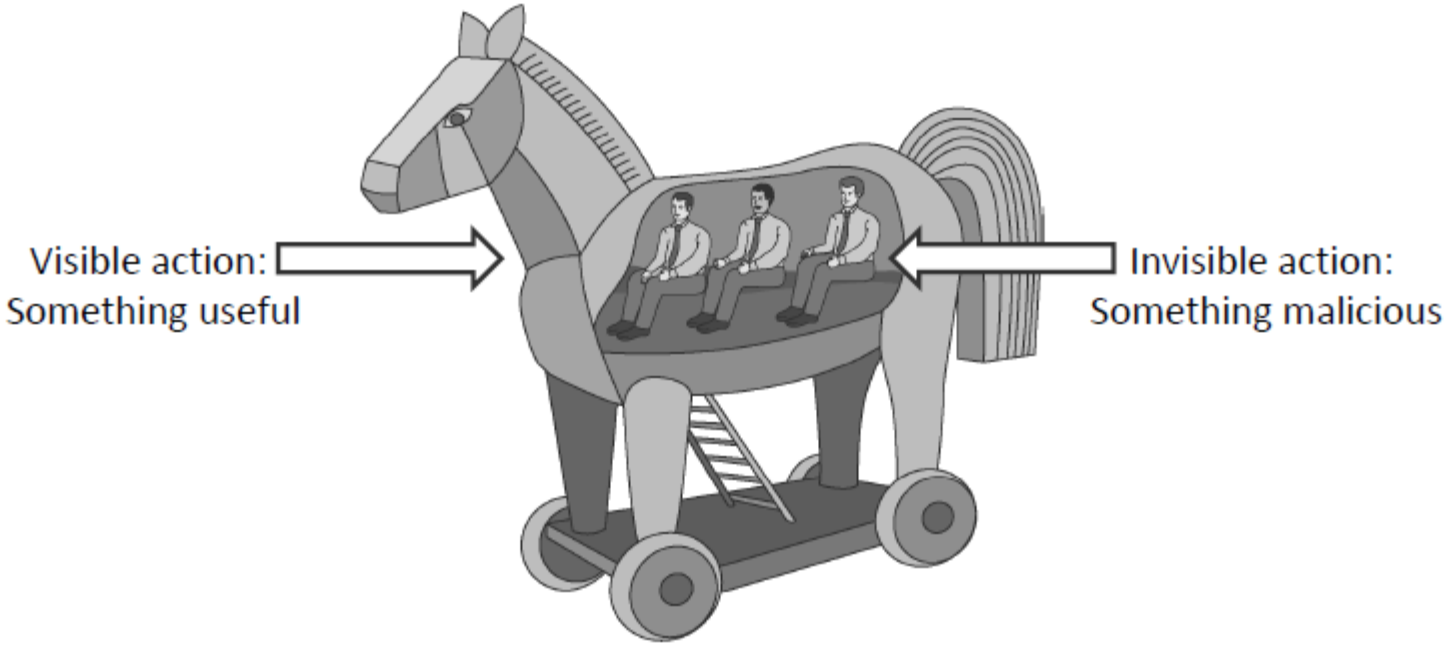
- First Worm program was in 1988, by Robert Morris, a student at that time, written for research purposes
- No malicious payload, just replicating itself
- Propagation by exploiting for example a buffer overflow in a networking program
- Checking if machine was already infected, but not trusting results 1/7 of times, and replicating anyway in such case
- Ended up many running many copies on same machine, consuming CPU resources, becoming an involuntary DOS attack
- Spread on 10% of all machines on internet at that time…
- Large unintended damage, ie wasted time in cleaning the infected machines
- Morris was convicted: 3 years if probation, 400 hours of community service, plus fine…

Other Kinds of Malware

Trojans, Rootkits, Botnets, etc.

Trojan Horses

- Reference to a Greek myth in *Aeneid*
- Program that does something useful (eg a music player), but also hide malicious code (eg steal your passwords / secrets)



Rootkits

- Malware that is installed at administrative level (ie, “root” user in Unix) of the OS, and alter the OS itself
- Very difficult to detect, because altering the functionalities of OS
- Might need to wipe out the hard-drive, and re-install the OS
- **Sony** 2005: most famous case, where Sony had rootkit malware in their CDs to alter Windows OS to install secret anti-copy protection software

Zero-Day Attacks

- A virus/worm to propagate needs to exploit an existing vulnerability in software (eg buffer overflow)
- Software can have bugs and security “holes” (eg, a buffer overflow vulnerability) not known to its developers
- Once a hole/bug is detected, the software can be *patched,* and a new release distributed
  - And that is why it is important to update software…
- *A “Zero-Day”* attach is an attack that exploits a vulnerability that is not known yet

Botnets

- Once machine infected with a worm, can be controlled remotely
- *Zombie*: an infected machine, which can receive commands from a *bot herder*
- *Botnets*: there can networks of millions of compromised machines (typically IoT devices) that can be controlled by a herder
- Uses: massive DDOS (distributed-denial-of-service) attacks and sending spam emails

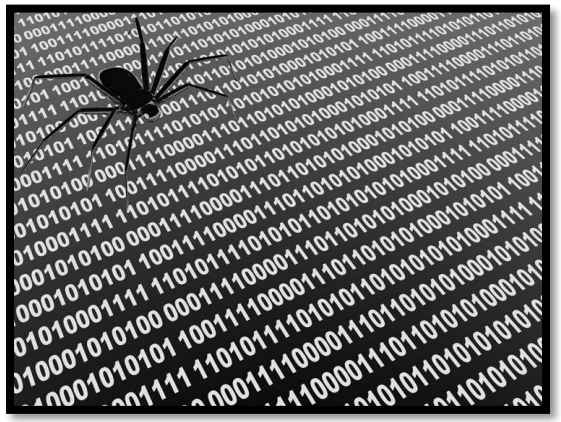
Adware

- Privacy-invasive software
- Display advertisements *without* user consent
- Eg, typically as pop-up message
- Can hide, and show pop-ups like they were coming from browser

Spyware

- Privacy-invasive software that collects data from user and send it to a malicious third-party
- *Keylogging*: record actions on keyboard, eg typing of passwords
- *Screen Capturing*: take snapshots of screen
- Detection: user would likely not notice the presence of spyware, as only direct negative effect would be just some CPU overhead of spyware running in background

Spyware malware



1. Infection of user machine

User

2. Spyware process collecting info like keystrokes and screenshots

3. Spyware process periodically sends data to one of hacker's servers

Spyware data collection server

2009-2011

- In 2009 *SpyEye* appeared, a "click-and-drag" programming tool for creating malware spyware that *steals bank information*
- You could buy it for 7000 Kroner
- Targeted attacks against Nordea and DnB NOR
  - https://www.dagbladet.no/nyheter/slik-tapper-de-pc-en-din-for-informasjon/63860991
- Several cases of stolen money from accounts
- 2011 those banks make public announcements on media to warn customers
- That’s also a reason for two-factor authentication and hardware one-time code devices

JavaScript Crypto Miners

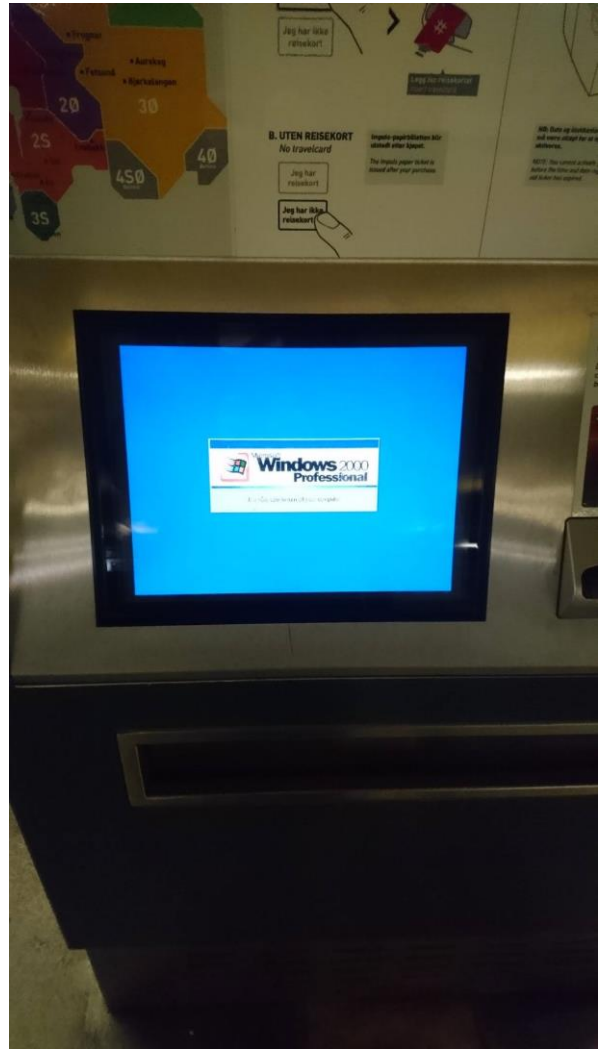
- Type of malware becoming more common from 2017, since value of cryptocurrencies (eg Bitcoins) has soared
- JavaScript code in malicious pages is loaded and executed, like any other legit JS code
- Consume CPUs of users to mine cryptocurrencies
- Negative effects: computer slower, higher electricity consumption, battery drain, etc

Countermeasures

- Install an antivirus
  - Not only for Windows, but Mac also…
  - Not just for viruses, but also for worms, Trojans, etc
- Keep your OS software updated
  - as security holes get patched…
- Eg, *never open email attachments that are programs*, or activate macros in Word documents
- Eg, beware of P2P networks, as often containing worms…
- Common sense: in general, never trust *unsolicited* files sent to you, *even from friends*
  - their machines could had been compromised

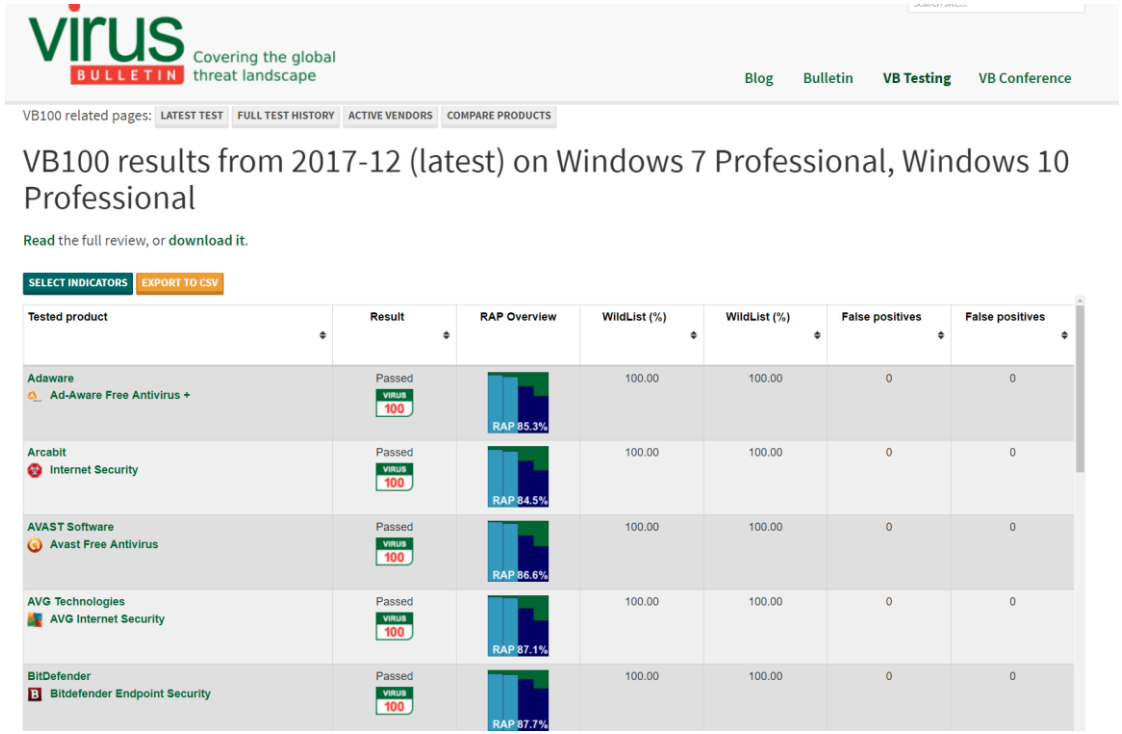
OS Updates Done Wrong…

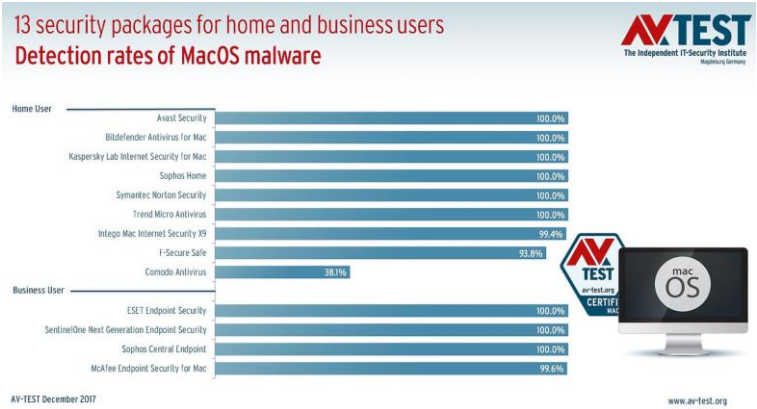
- **2018**, picture from T-Bane in Grønland Torg
- Machine where you buy metro tickets
- **Windows 2000** has not been supported since **2010**…
  - https://blogs.technet.microsoft.com/education/2009/11/10/windows-2000-end-of-life/
- Think about it next time you put a credit card in one of those machines…
- If once you graduate then get a job at *Ruter*, please help them improving their security practices…



Many anti-virus vendors out there…

- Not just Windows, also Mac…
  - https://www.virusbulletin.com/testing/results/latest/vb100-antimalware
  - https://www.av-test.org/en/news/news-single-view/put-to-the-test-antivirus-solutions-for-macos-sierra/
  - https://www.av-test.org/en/antivirus/home-windows/





JavaScript

Main Characteristics

- **Interpreted**: you do not need to compile it (eg, in contrast to Java which is compiled down to bytecode)
  - Note: for performance reasons, the *runtime* (eg a browser like Chrome) will compile JS *on the fly* into machine code
- **Dynamically Typed**: when declaring variables, no need to specify the type, eg String or Numeric, and can reassign to different types
- **Weakly Typed**: you can use operators like “+” and “-” on different types (eg arrays and strings) without throwing errors

Interpreted

- Can just provide source code directly to the browser
- Can be directly inside HTML, or in separated ".js" files imported like any other resource (CSS, images, etc.)
- Note: current practice is to use *transpilation* steps
  - eg, using build tools like NPM
  - bundle dependencies like libraries (React/Angular/Vue/etc.)
  - transformations to support old browsers
  - enabling typing with TypeScript
  - etc.

Dynamically Typed

- **var x = 1;**
  - declare a variable called **x** with a numeric value equal to **1**
  - note we did not need to specify the “numeric” type
- **var x = 1; var x = “a”;**
  - **x** contains a string in the end. So, we changed the type from numeric to string
- **x = 1**
  - the “**var**” and “**;**” could be omitted, but you should NOT omit them
  - “**var**”: makes a local variable, otherwise is global scope (which is *bad*)
  - omitting “**;**” can lead to subtle bugs…

Weakly Typed

- A string plus a number? Concatenation
  - **“a” + 1** becomes **“a1”**
- A string minus a number? Result is not a number…
  - **“a” – 1** becomes **NaN**
- An empty object plus an empty array? Numeric 0…
  - **{} + []** becomes **0**
- Other dynamically typed languages (eg, *Python*) would throw an exception at runtime
  - They are called *Strongly Typed*
- Statically typed languages (eg, *Java*) would not even *compile*
  - with the only *exception* of “+” on String objects

For equality, use “===“ and not “==”

- **false == 0**
  - result is **true**, ie, boolean **false** is equivalent to numeric **0**, as the **0** gets transformed into a boolean to compare it with **false**
- **false === 0**
  - result is **false**, as a boolean value is not equal to a numeric value
- **0 == []**
  - surprisingly, that is true in JS, ie the numeric **0** is equal to an empty array
  - plenty of these hilarious cases, see https://dorey.github.io/JavaScript-Equality-Table/
- For negation, use **!==** instead of **!=**

Code Comments

- To document software, typical case of writing comments directly in the source code
- JS uses similar syntax to other languages (eg Java)
- Single-line comment: **//**
- Multi-line comment: started with **/*** and then closed with ***/**

Function Declaration

- **foo = function(){ return 1;}**
  - calling **foo()** will return value **1**
- **add = function(x,y){return x+y;}**
  - calling **add(1,2)** will return **3**
  - calling **add(“a”, “b”)** will return **“ab”**

DOM Manipulation

- Document Object Model (DOM): object representation of the displayed HTML
- One of the main reasons to use JS is to manipulate the DOM, ie altering what is displayed to the user
- To access the DOM, JS can refer to the object called "**document**"
- Call methods on **document** to retrieve object representations of the DOM

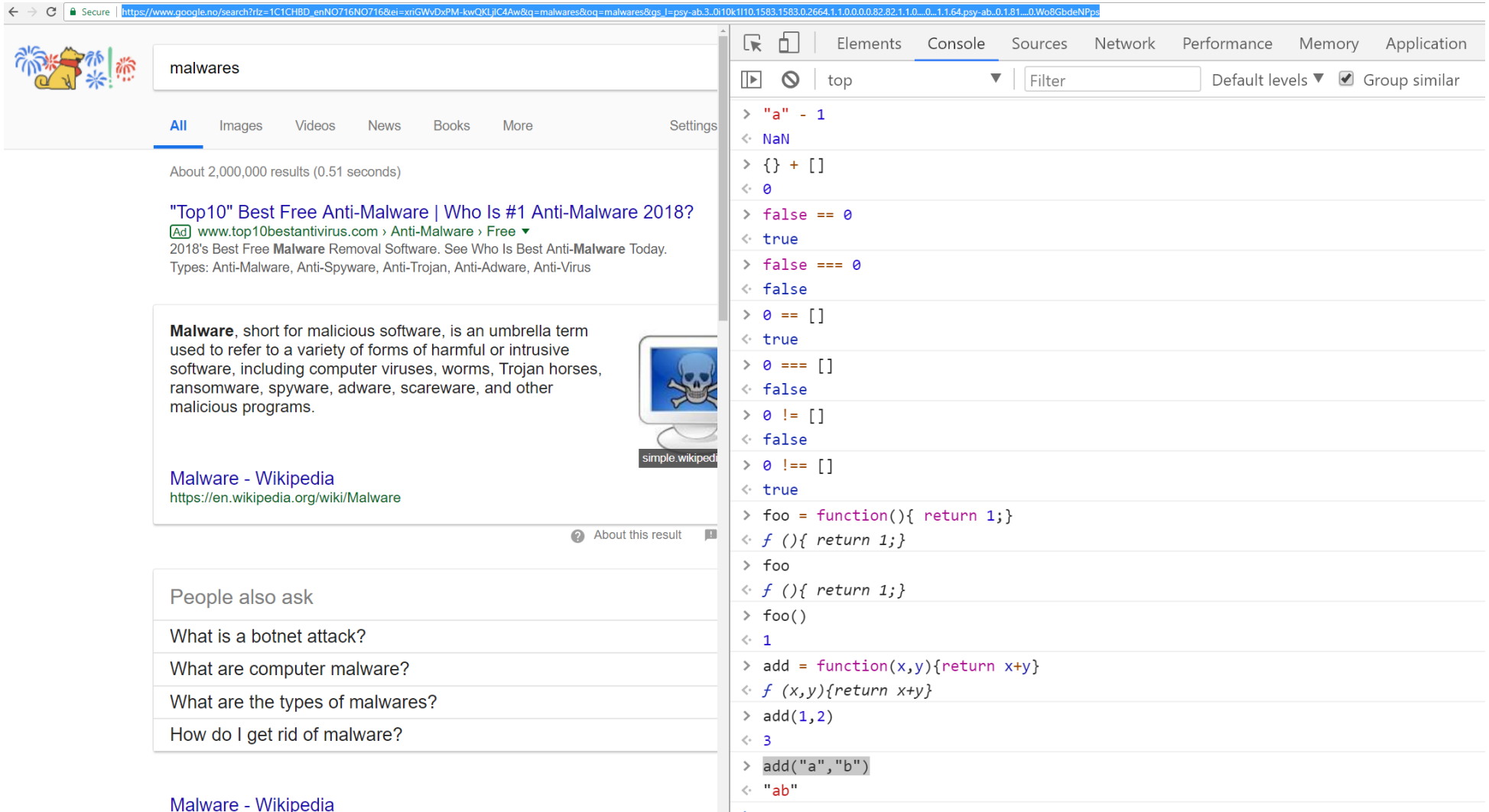
```
clearText = function(){

    var textArea = document.getElementById("textId");
    var resultArea = document.getElementById("resultId");

    textArea.value = '';
    resultArea.value = '';
};
```

- Easiest way to retrieve DOM objects is by *id*
- The id needs to be set as HTML attribute, e.g. **<textarea id="textId"></textarea>**

JS Interactions

- There are different ways to execute JS in a page
- One simple approach is to directly register *event handlers* on the HTML tags
  - ** Clear **
  - when user on browser clicks on that button, the JS function “clearText()” is going to be executed
- Event handlers:
  - *onclick*, *onchange*, *onmouseover*, *onmouseout*, *onkeydown*, etc.
  - see for example https://www.w3schools.com/js/js_events.asp

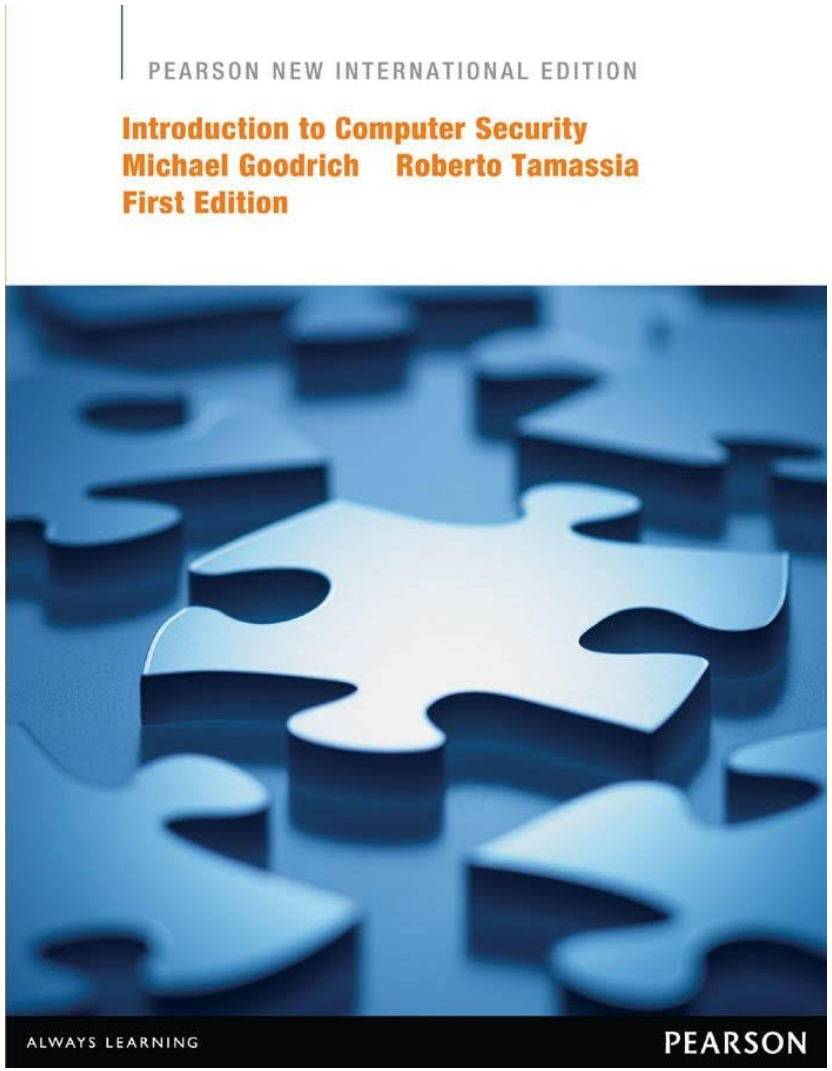
JS Console, from Chrome Developer Tools, useful for debugging and learning by running custom JS directly on page



JS Koans

- Koans are sets of exercises to learn different languages, where you need to fix a series of test cases that fail
- People have made different Koans for JS, and we will use https://github.com/liammclennan/JavaScript-Koans
  - Note: there are quite a few topics there that most likely we will not need in this course, like data structures (stacks and queues), prototypes and regular expressions
- **DEMO**

For Next Week



- Book pages: 174-214
- Note: when I tell you to **study** some specific pages in the book, it would be good if you also *read* the other pages in the same chapter at least once
- Exercises for Lesson 4 on GitHub repository