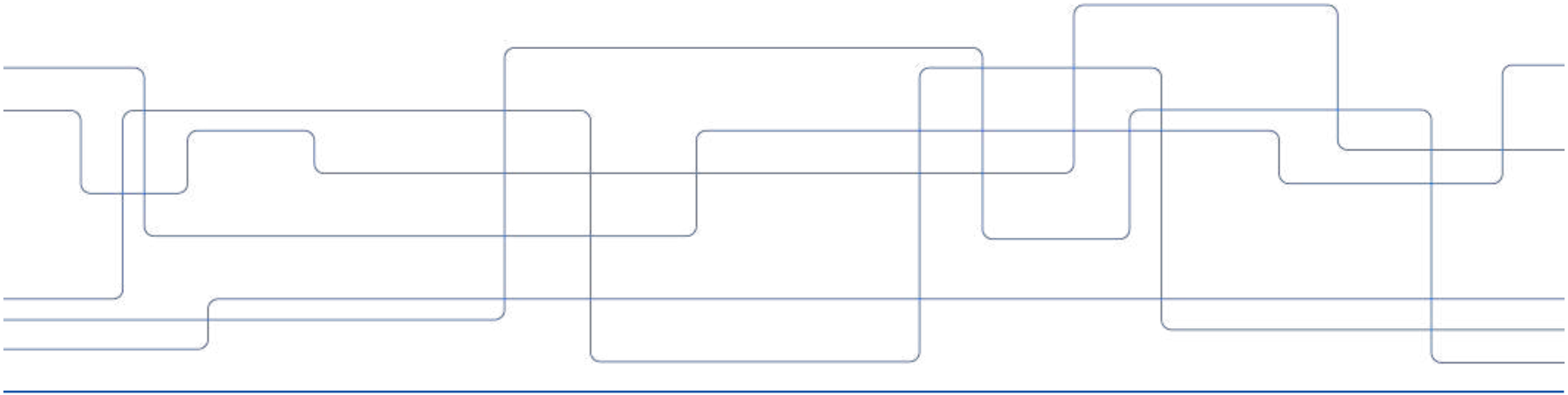
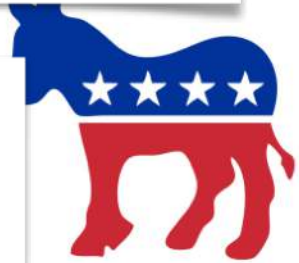
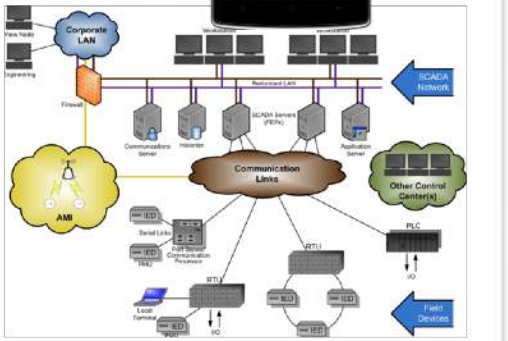
Grundorsakerna till sårbarheter i programvara

Professor Pontus Johnson






Corporate LAN
SCADA Network
Communication Links
AMI
Other Control Center(s)
PLC
Field Devices


What Happened to My Computer?
Can I Recover My Files?
How Do I Pay?
bitcoin
Check Payment
Decrypt

CYBER THREAT ACTOR	MOTIVATION
NATION-STATES	GEOPOLITICAL
CYBERCRIMINALS	PROFIT
HACKTIVISTS	IDEOLOGICAL
TERRORIST GROUPS	IDEOLOGICAL VIOLENCE
THRILL-SEEKERS	SATISFACTION
INSIDER THREATS	DISCONTENT

Belfer Center
National Cyber Power Index 2020 "Top 10"

			Specific Rankings	
#	Country	Overall score	Capability	Intent
1	United States	50.24	1	2
2	China	41.47	2	1
3	United Kingdom	35.57	3	3
4	Russia	28.38	10	4
5	Netherlands	24.18	9	5
6	France	23.43	5	11
7	Germany	22.42	4	12
8	Canada	21.50	11	9
9	Japan	21.03	8	14
10	Australia	20.04	16	8

2020

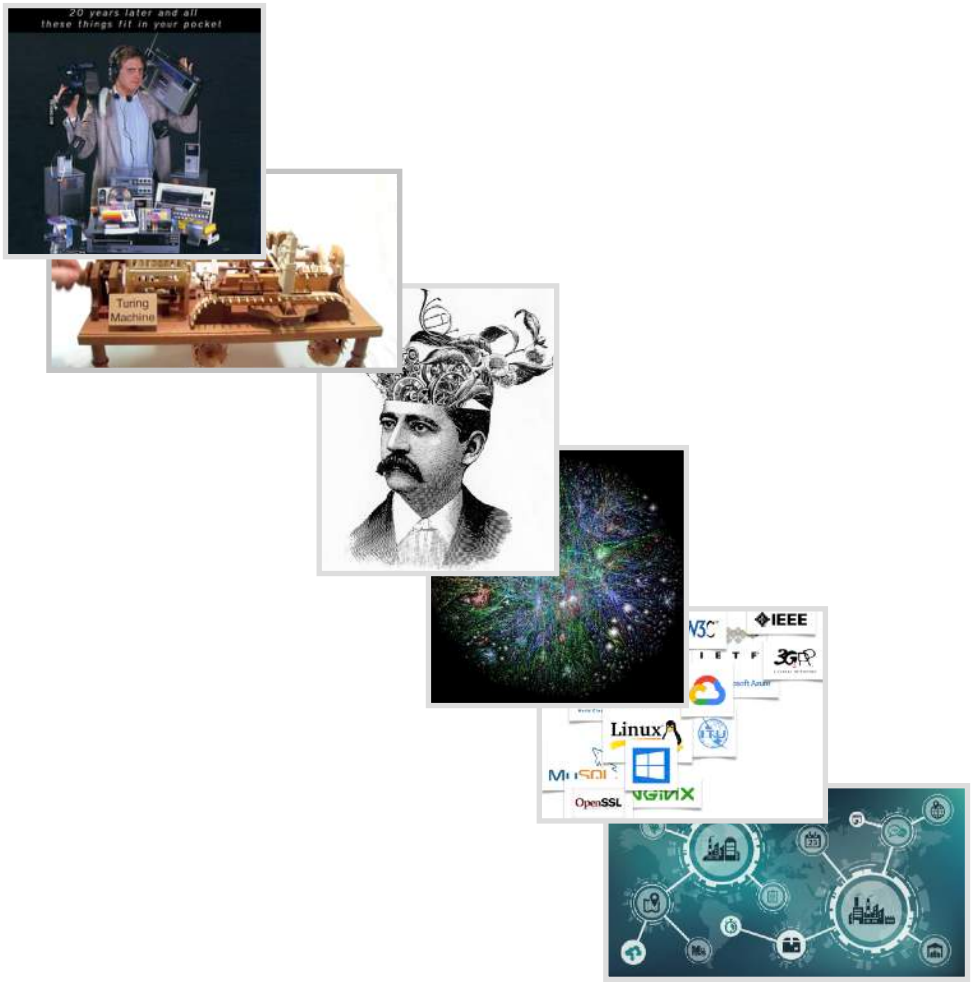
Cybersäkerhet i Sverige

– Rekommenderade säkerhetsåtgärder

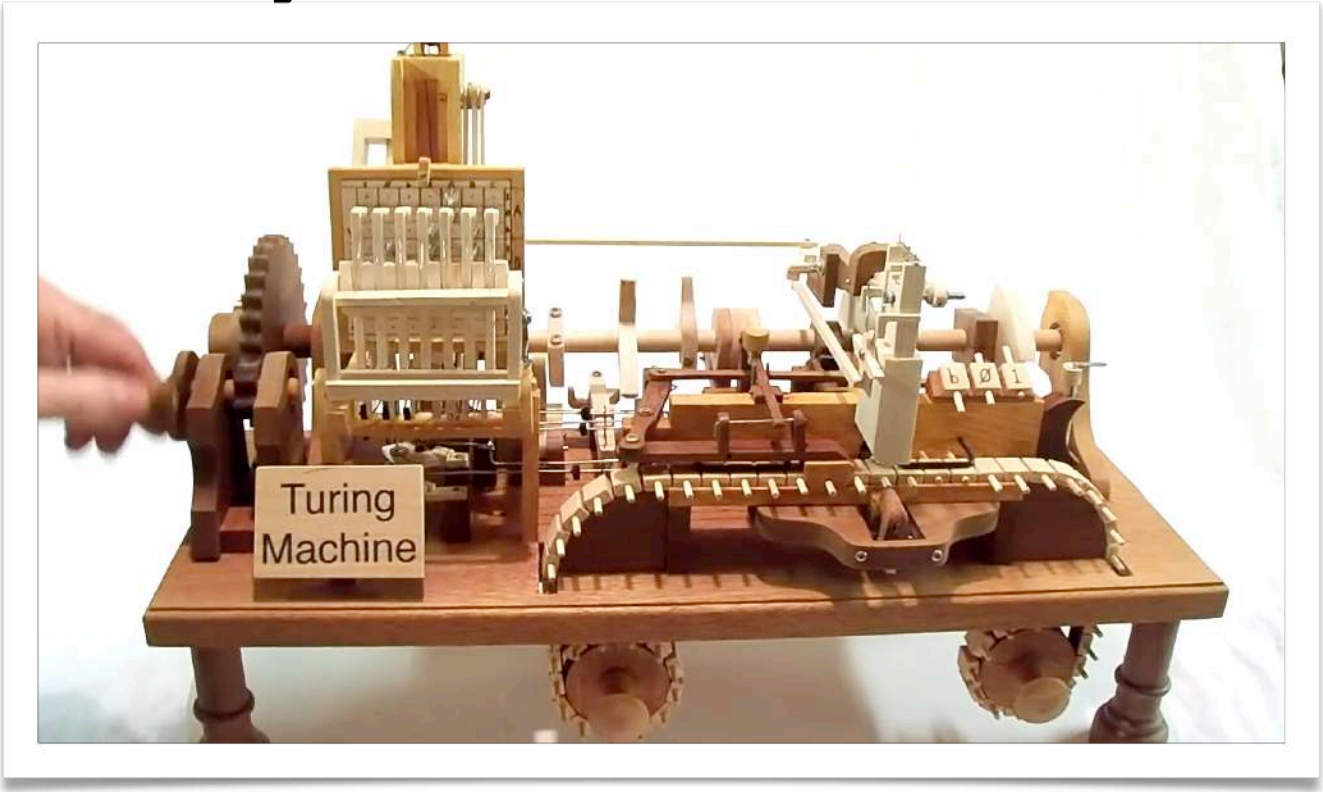
1. Installera säkerhetsuppdateringar så fort det går
2. Förvalta behörigheter och använd starka autentiseringsfunktioner
3. Begränsa och skydda användningen av systemadministrativa behörigheter
4. Inaktivera oanvända tjänster och protokoll (härda systemen)
5. Gör säkerhetskopior och testa om informationen går att läsa tillbaka
6. Tillåt endast godkänd utrustning i nätverket
7. Säkerställ att endast godkänd mjukvara får köras (vitlistning)
8. Segmentera nätverken och filtrera trafiken mellan segmenten
9. Uppgradera hård- och mjukvara
10. Säkerställ en förmåga att upptäcka säkerhetshändelser

Root causes

- Universality
- Determinism
- Cognitive complexity
- Exposure
- Monoculture
- The supply chain



Universality



Turing completeness - all computers can simulate each other

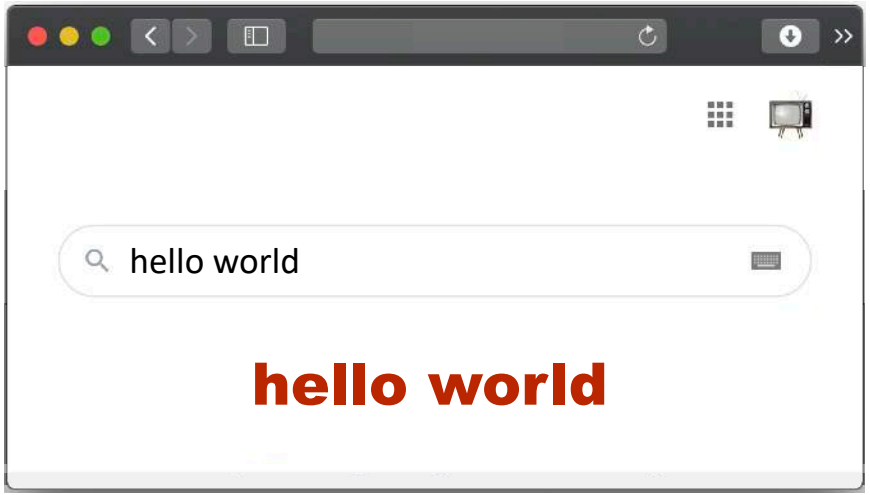
Root cause: Universality

A smartphone can, and does, simulate any information-processing machine

Universality can be subverted

A universal machine can be repurposed into any other machine

The interchangeability of program and data



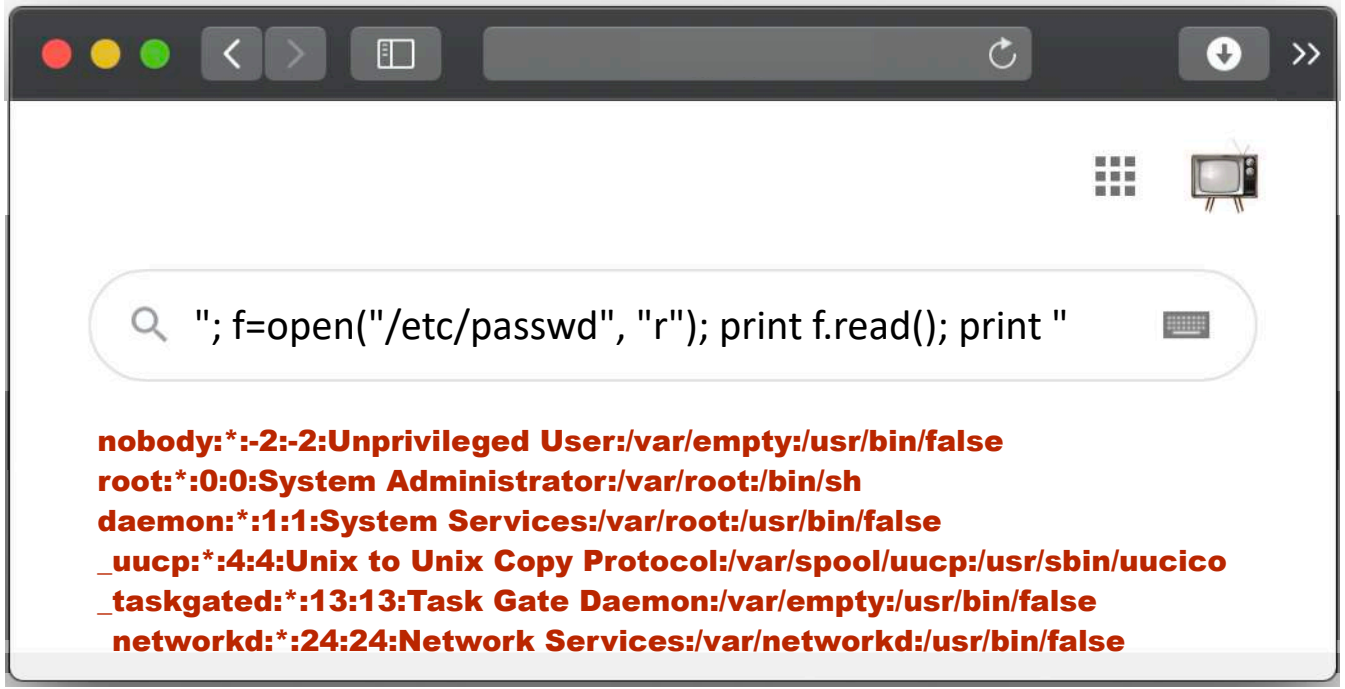
Program vulnerable.py

```
import sys
my_data = sys.argv[1]
my_program = 'print "' + my_data + '"'
print my_program
exec(my_program)
```

Intended use

```
% python vulnerable.py 'hello world'
print "hello world"
hello world
```

Data becomes code

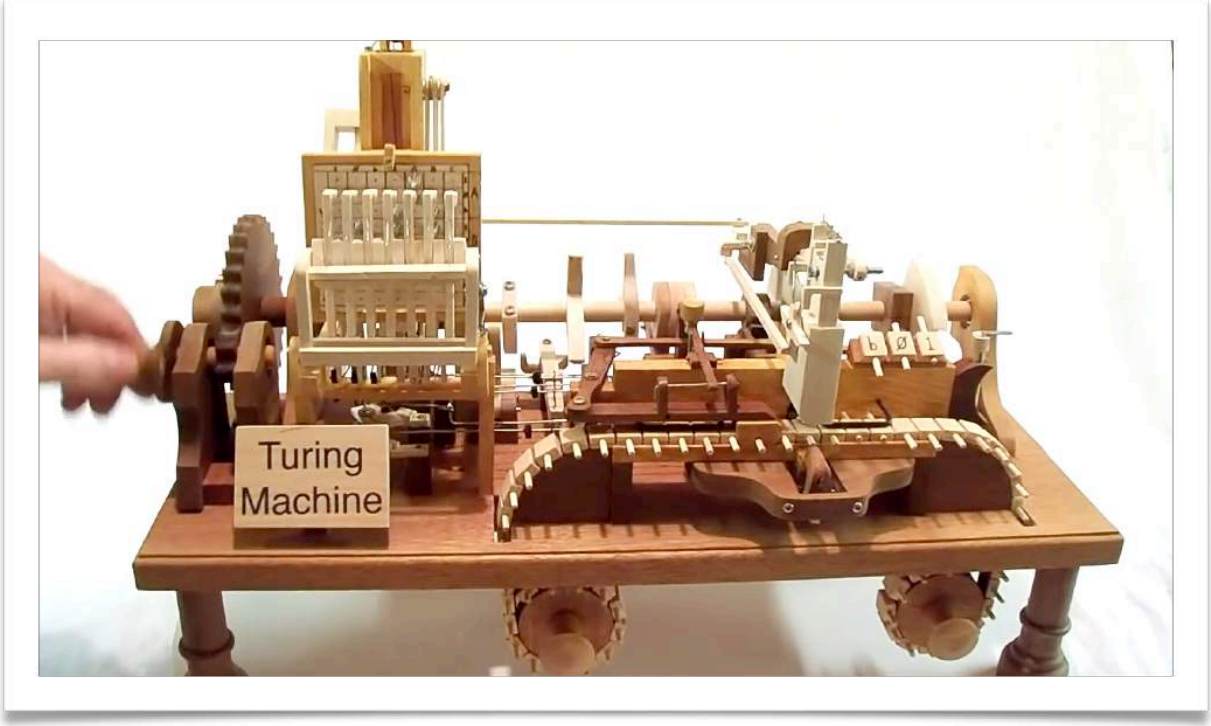



```
print ""; f=open("/etc/passwd", "r"); print f.read(); print ""
```

Is there a fundamental solution to this problem?

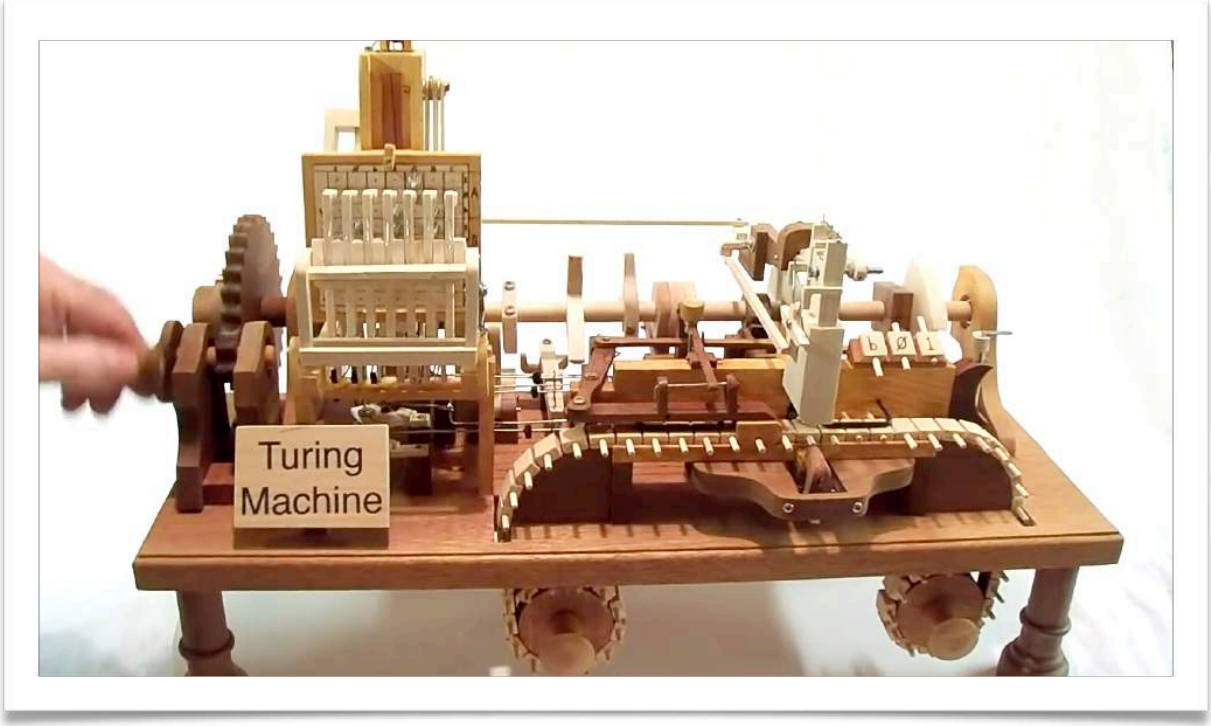
No solution.

We want the Universal Machine.



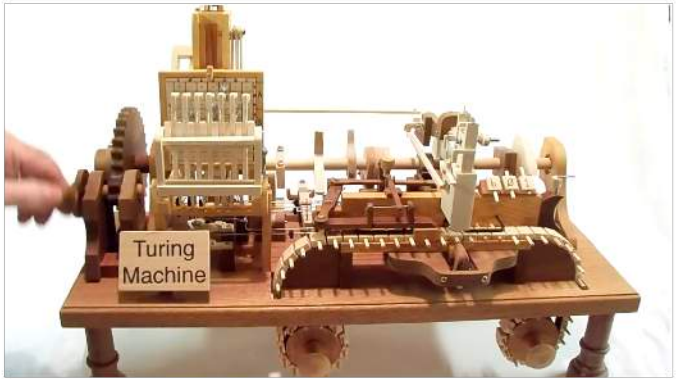
Root cause: Determinism

All computers we use today are *Deterministic*.



Determinism

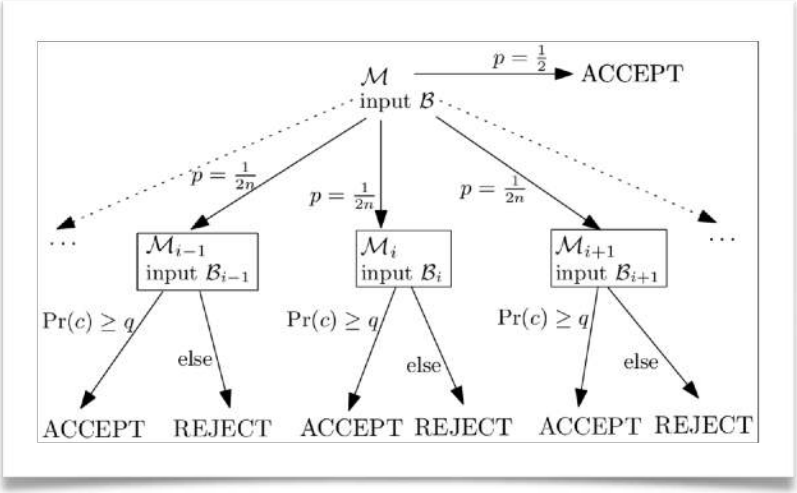
- Deterministic Turing machine
- A hacker can try and try again
- Once vulnerable, always vulnerable

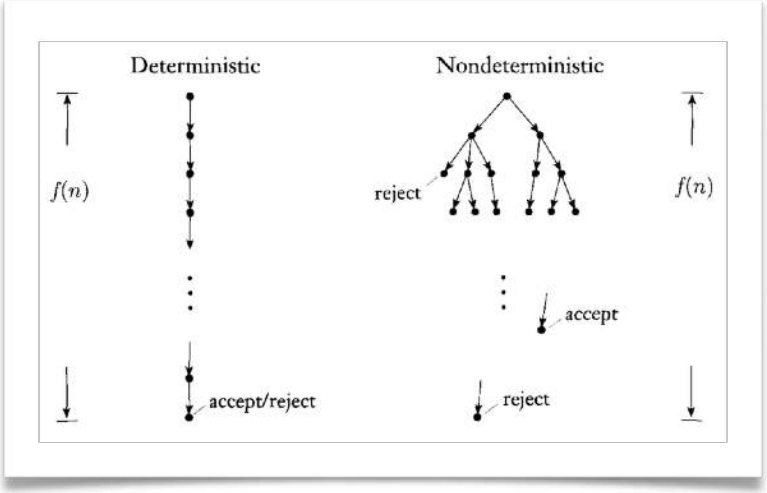


fool me once, shame on you;
fool me twice, shame on me

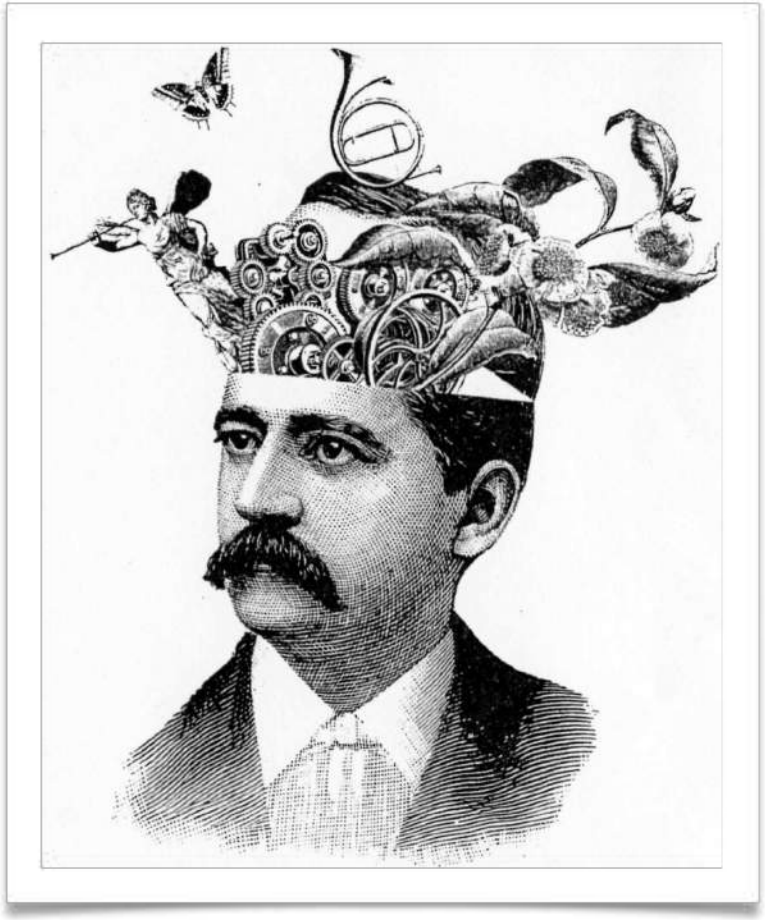
Is there a fundamental solution to this problem?

- Not likely - Nondeterministic Turing machines and probabilistic Turing machines.





Root cause: Cognitive complexity



Sad DNS

- En sårbarhet som ingen upptäckte på 12 år, trots att den ursprungliga sårbarheten (Kaminsky) var jättestor
- Ingen kom på Sad DNS för att det var alltför kognitivt komplext att kombinera ICMP rate limiting, m.m.

It is easy to create complicated programming languages

Esoteric programming languages

Shakespeare

```
Romeo, a young man with a remarkable
patience.
Juliet, a likewise young woman of
remarkable grace.
Ophelia, a remarkable woman much in
dispute with Hamlet.

Act I: Hamlet's insults and flattery.
Scene I: The insulting of Romeo.
[Enter Hamlet and Romeo]
[...]
Hamlet:
 Thou art as sweet as the sum of the sum
of Romeo and his horse and his
 black cat! Speak thy mind!
[Exit Juliet]
[...]
Scene III: The praising of Ophelia.
[Enter Ophelia]
Hamlet:
```

BrainFuck

```
+++>++>>>+[>>,[>++++++<[[->]<<]<[>]>]>-[<<+
++++>>-[<<----->>-[->]<]]

<[<-<[<]+<+[>]<<+>->>>]<]<[<]>[-[>+++++
+<-]>[<+>-]+<<<+++>+>

  [-

    [<<+>->-

      [<<[-]>>-

        [<<++>+>-

          [<<-->->>+++<-

            [<<+>+>>--<-

              [<<->->-

                [<<++++>+>>+<-

                  [>-<-
```

It is difficult to create simple programming languages

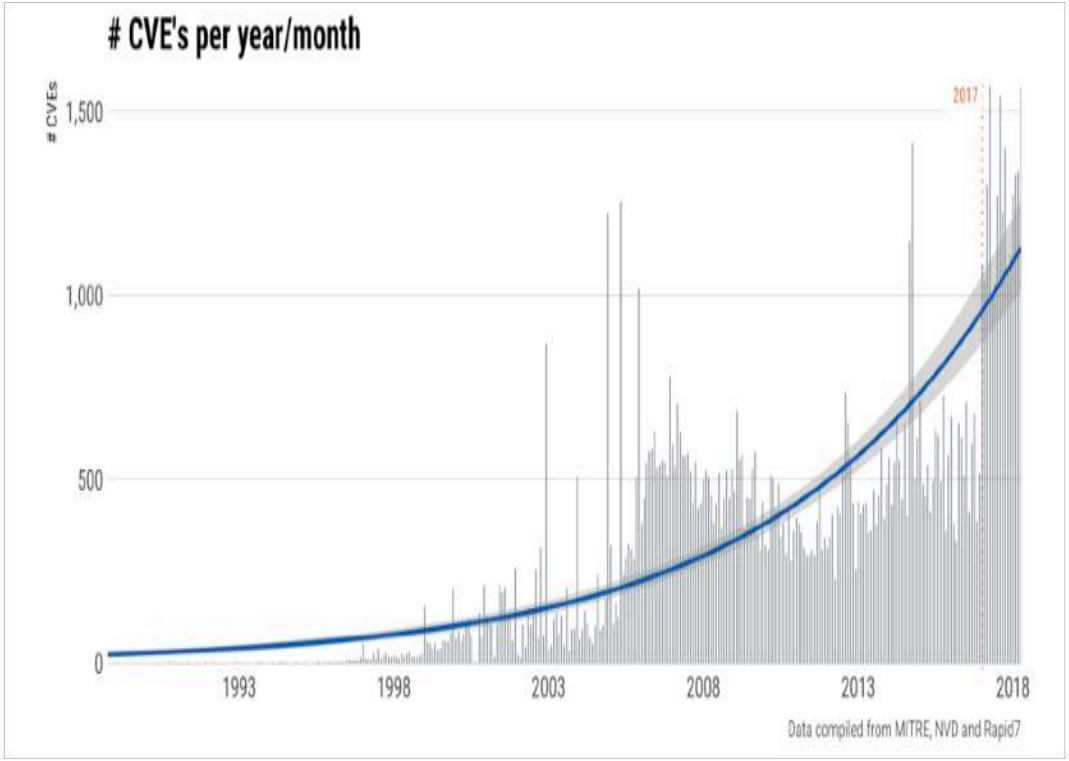
C programming language

```
#include <stdio.h>
int main()
{
    int n1, n2, i, g;

   scanf("%d %d", &n1, &n2);

    for(i=1; i <= n1 && i <= n2; ++i)
    {
       if(n1%i==0 && n2%i==0)
            g = i;
    }

    printf("%d %d: %d", n1, n2, g);

    return 0;
}
```

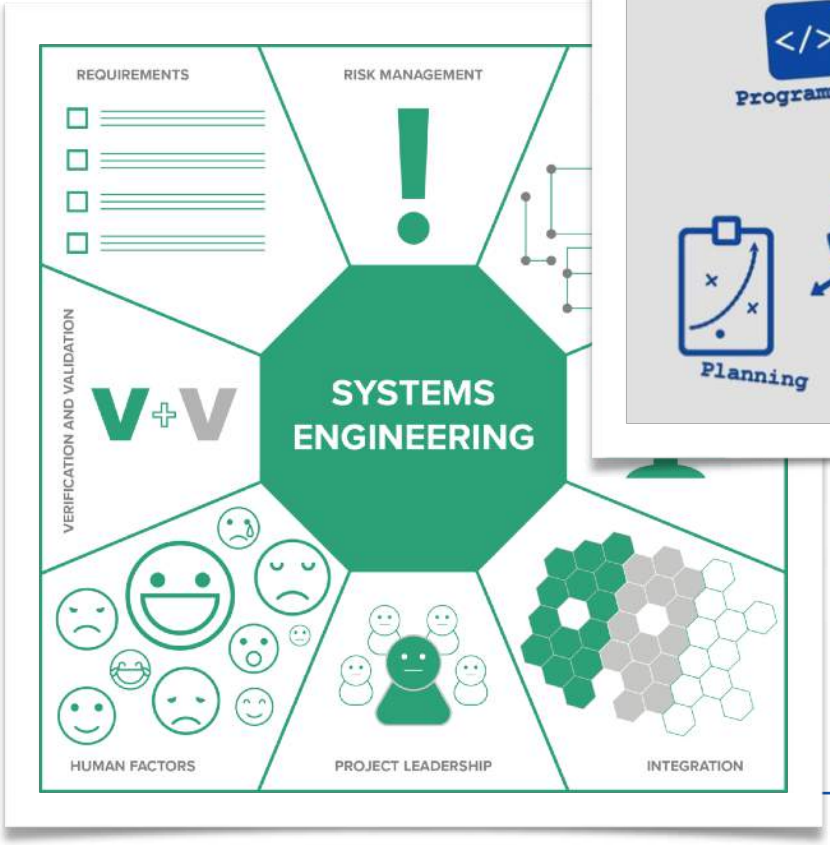
Not only programming languages, but all such abstractions:

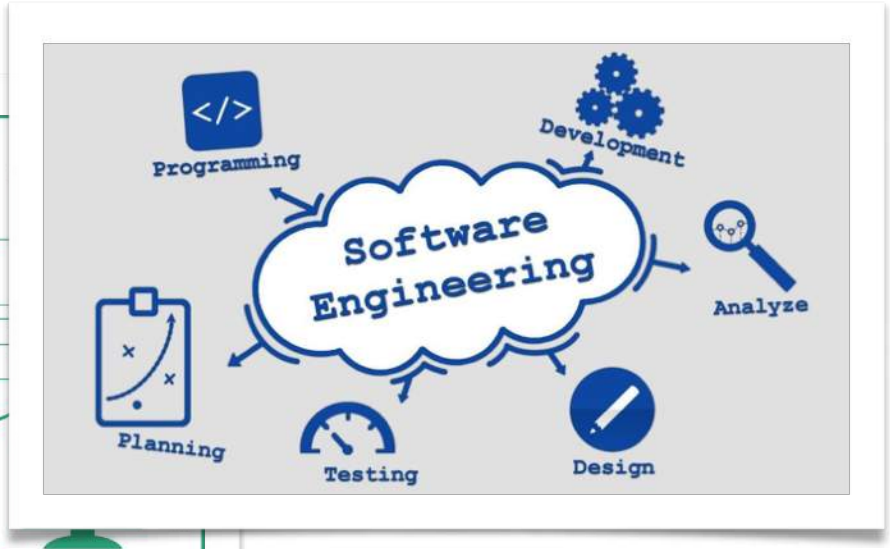
- OS APIs
- Library APIs
- Web APIs
- Instruction sets
- ...

Complexity is the enemy of security



Is there a fundamental solution to this problem?

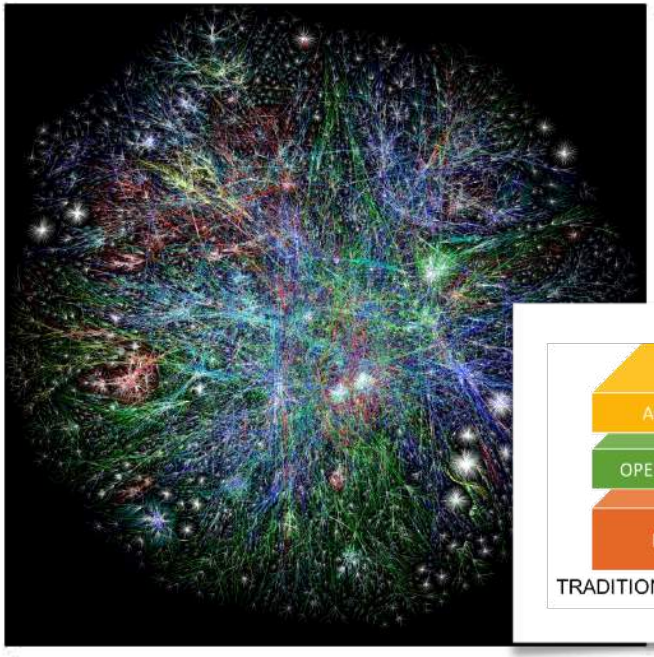




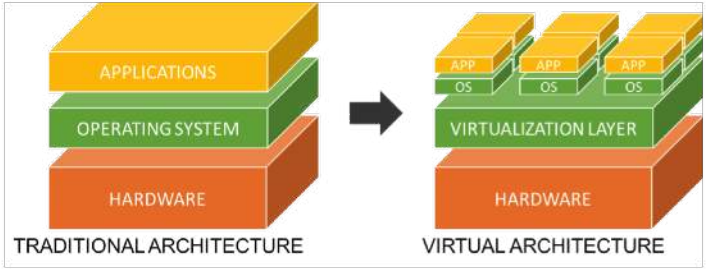
...but today's methods and tools are obviously not good enough

Root cause: Exposure

Physical access



The Internet



Virtualization

Current solutions

Network segmentation, firewalls

Sandboxing, virtualization

Access control

Cryptography

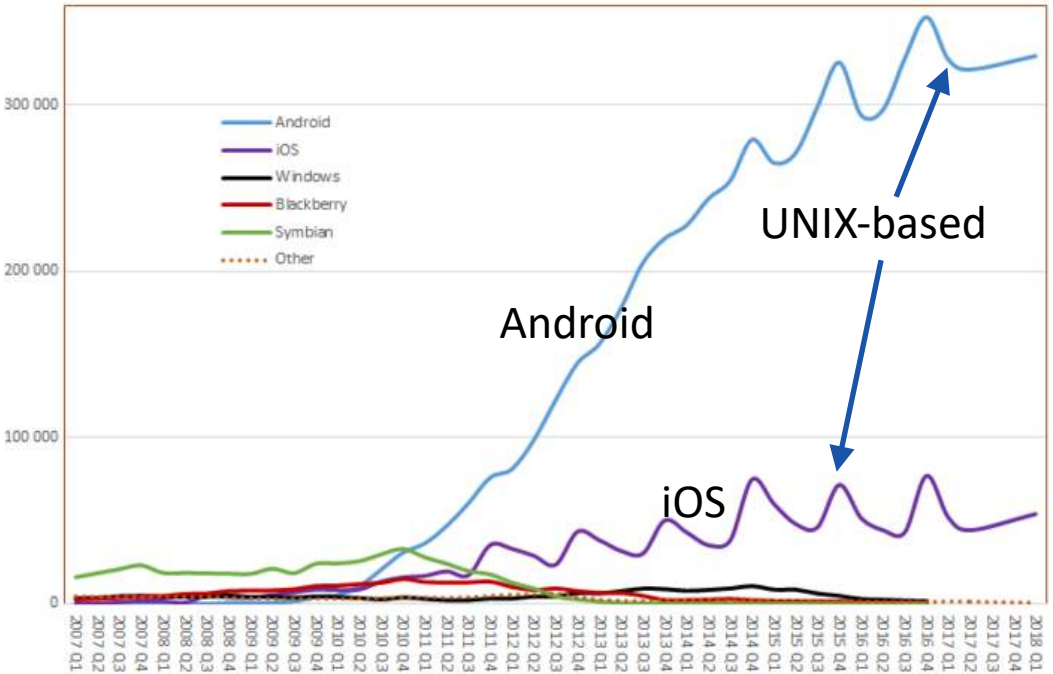
Fundamental solutions to this problem

Trade-off between productivity and security. No solution, only balance.

...but the cost of lost productivity is high.

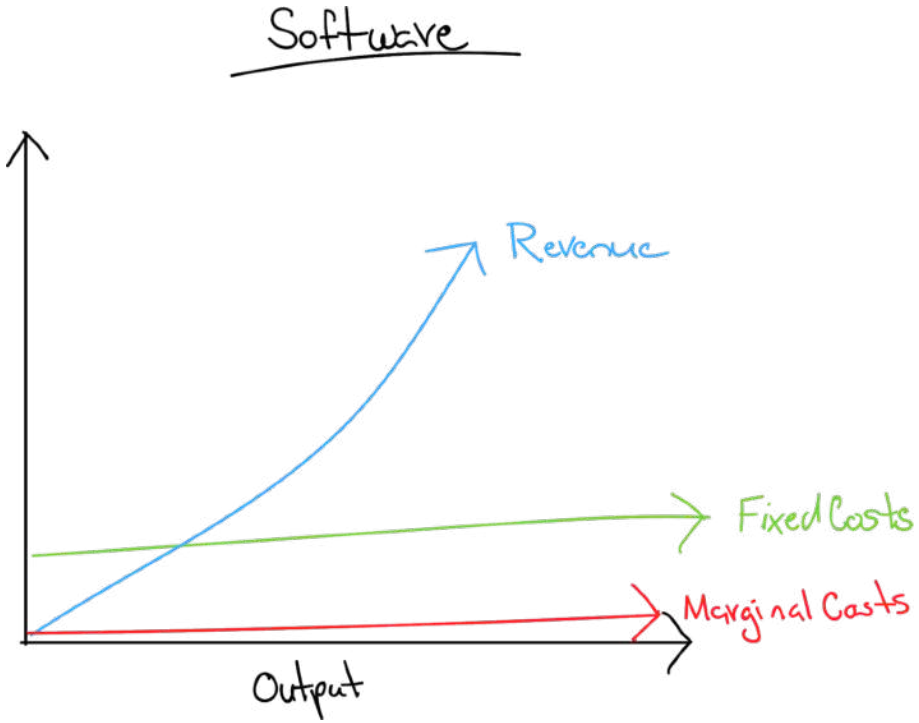
Security vs usability

Root cause: Monoculture



World-Wide Smartphone Sales (Thousands of Units)

Marginal cost of software is zero



Is there a fundamental solution to this problem?

Trade-off between productivity and security.

No solution, only balance.

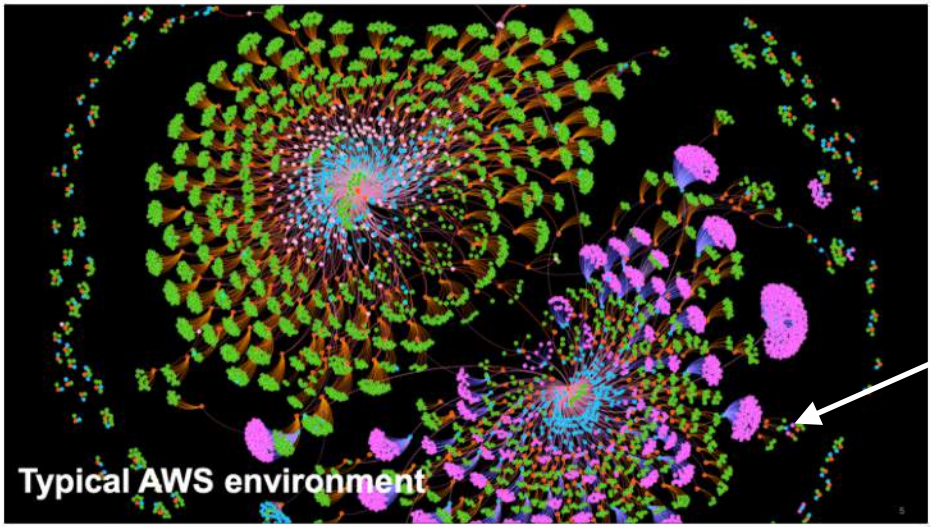
...but the cost of lost productivity is very high.

Root cause:
The supply chain

Linux distrubution (Kali 2020.1) installed packages

Kernel is here

21,000 developers (as of 2020)



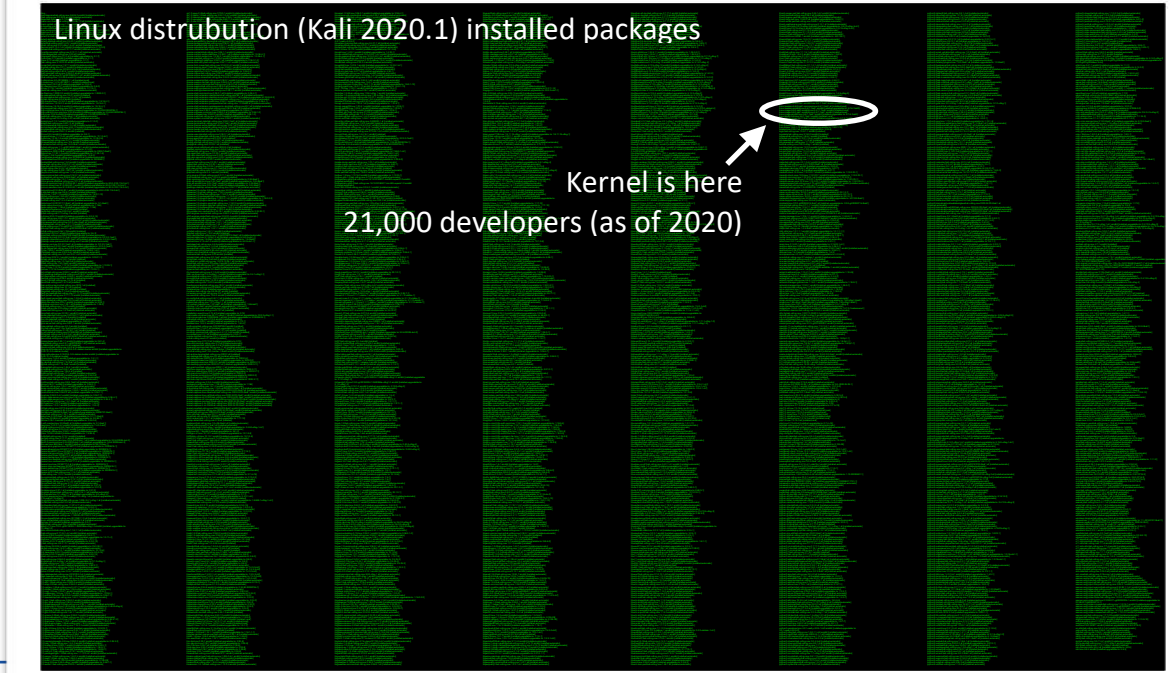
One Linux installation

Is there a fundamental solution to this problem?

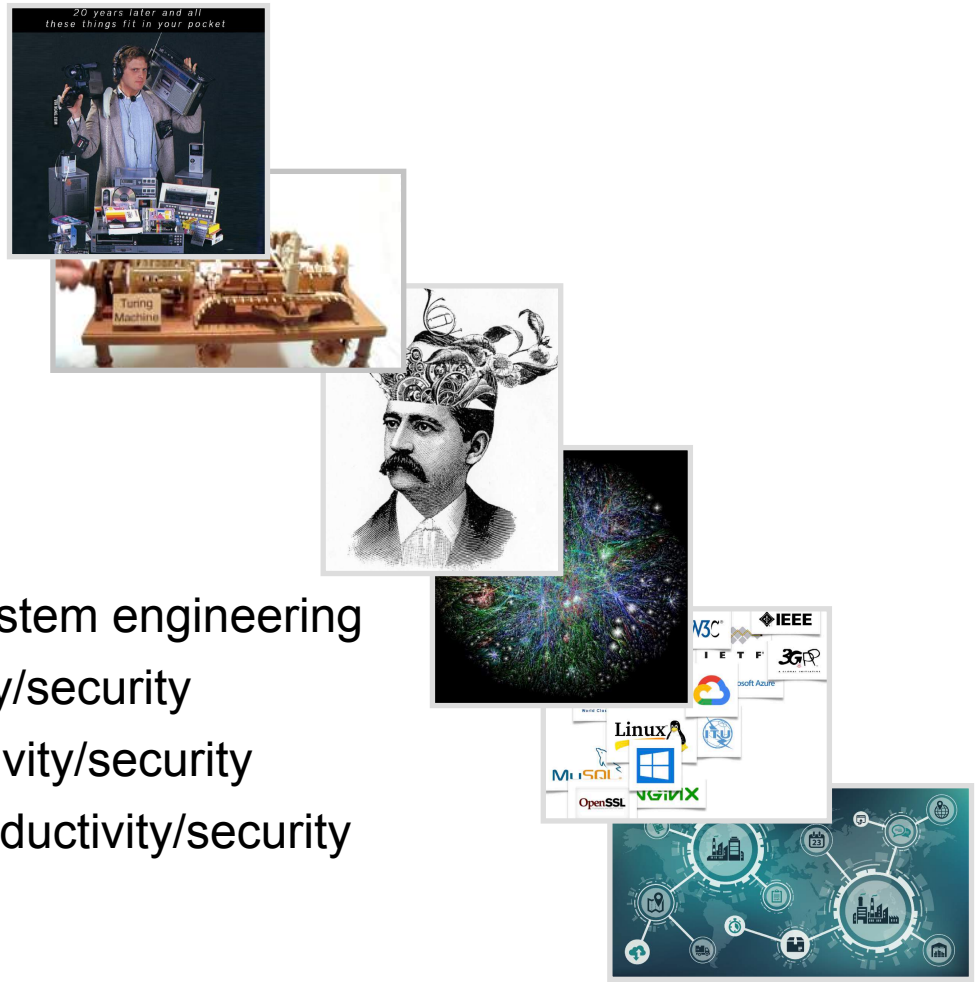
Trade-off between productivity and security.

No solution, only balance.

...but the cost of lost productivity is very high.



Root causes and solutions



- Universality - No solution
- Determinism - No solution
- Cognitive complexity - Much better system engineering
- Exposure - costly trade-off productivity/security
- Monoculture - costly trade-off productivity/security
- The supply chain - costly trade-off productivity/security