Mutter aller Sicherheitslücken
Arbitrary Code Execution in the Universal Turing Machine

Prof Pontus Johnson
KTH Royal Institute of Technology

"~~Engineering~~ is the closest thing to magic that exists in the world."

Elon Musk

Arbitrary code execution

Arbitrary code execution

2019 CWE Top 25 Most Dangerous Software Errors

Rank	ID	Name
[1]	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer
[2]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
[3]	CWE-20	Improper Input Validation
[4]	CWE-200	Information Exposure
[5]	CWE-125	Out-of-bounds Read
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
[7]	CWE-416	Use After Free
[8]	CWE-190	Integer Overflow or Wraparound
[9]	CWE-352	Cross-Site Request Forgery (CSRF)
[10]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
[11]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
[12]	CWE-787	Out-of-bounds Write
[13]	CWE-287	Improper Authentication
[14]	CWE-476	NULL Pointer Dereference
[15]	CWE-732	Incorrect Permission Assignment for Critical Resource
[16]	CWE-434	Unrestricted Upload of File with Dangerous Type
[17]	CWE-611	Improper Restriction of XML External Entity Reference
[18]	CWE-94	Improper Control of Generation of Code ('Code Injection')
[19]	CWE-798	Use of Hard-coded Credentials
[20]	CWE-400	Uncontrolled Resource Consumption
[21]	CWE-772	Missing Release of Resource after Effective Lifetime
[22]	CWE-426	Untrusted Search Path
[23]	CWE-502	Deserialization of Untrusted Data
[24]	CWE-269	Improper Privilege Management
[25]	CWE-295	Improper Certificate Validation

Example arbitrary code execution SQL injection

Program

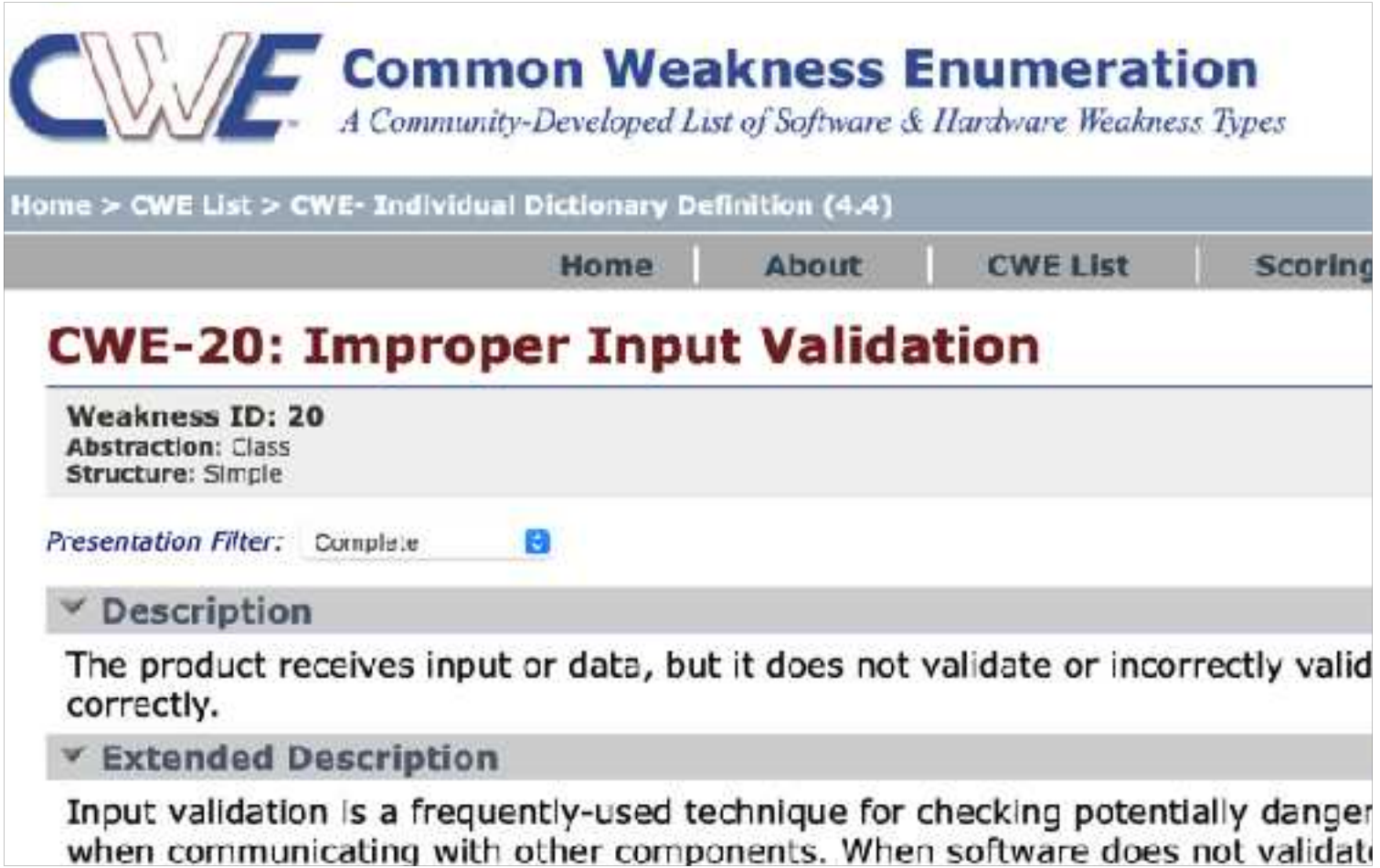
```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

Malicious input

```
105; DROP TABLE Users
```

Result

```
SELECT * FROM Users WHERE UserId = 105; DROP TABLE Users;
```


CWE
Common Weakness Enumeration
A Community-Developed List of Software & Hardware Weakness Types
Home > CWE List > CWE- Individual Dictionary Definition (4.4)
Home
About
CWE List
Scoring
CWE-20: Improper Input Validation
Weakness ID: 20
Abstraction: Class
Structure: Simple
Presentation Filter:
Complete
Description
The product receives input or data, but it does not validate or incorrectly valid
correctly.
Extended Description
Input validation is a frequently-used technique for checking potentially danger
when communicating with other components. When software does not validat

Entscheidungsproblem

Is there an algorithm that will take a formal language, and a logical statement in that language, and that will output "True" or "False", depending on the truth value of the statement?

David Hilbert

1928

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope

The halting problem

From a description of an arbitrary computer program and an input, determine whether the program will finish running, or continue to run forever.

Most cited papers citing [Turing 1936]

Subjects

Neural networks
Post-quantum cryptography
Economics
Psychology
Futurology
Complex systems
Algorithmic information theory
Quantum theory
Numerical analysis
Computational linguistics
Parallel computing
Theory of computation
Postmodernism
Computational complexity
Cellular automata
Automata theory
Formal languages

- AM Turing, Computing machinery and intelligence, 1950(15801)
- J Schmidhuber, Deep learning in neural networks: An overview, 2015 (12480)
- PW Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, 1999 (1
- N Georgescu-Roegen, The entropy law and the economic process, 2013 (9089)
- R Penrose, ND Mermin, The emperor's new mind: Concerning computers, minds, and the laws of physics, 1990 (815
- R Kurzweil, The singularity is near: When humans transcend biology, 2005 (7189)
- JH Holland, Hidden order: How adaptation builds complexity, 1996 (6797)
- M Li, P Vitányi, An introduction to Kolmogorov complexity and its applications, 1993 (6921)
- D Deutsch, Quantum theory, the Church–Turing principle and the universal quantum computer, 1985 (6162)
- NJ Higham, Accuracy and stability of numerical algorithms, 2002 (5580)
- D Jurafsky, Speech & language processing, 2000 (12209)
- H Gardner, The mind's new science: A history of the cognitive revolution, 1987 (6414)
- LG Valiant, A bridging model for parallel computation, 1990 (4930)
- JH Holland, Emergence: From chaos to order, 2000 (4914)
- R Penrose, Shadows of the Mind, 1994 (4765)
- M Sipser, Introduction to the Theory of Computation, 1996 (4501)
- P Cilliers, Complexity and postmodernism: Understanding complex systems, 2002 (4312)
- ML Minsky, Computation, 1967 (4438)
- R Rojas, Neural networks: a systematic introduction, 2013 (4370)
- S Wolfram, Statistical mechanics of cellular automata, 1983 (3964)

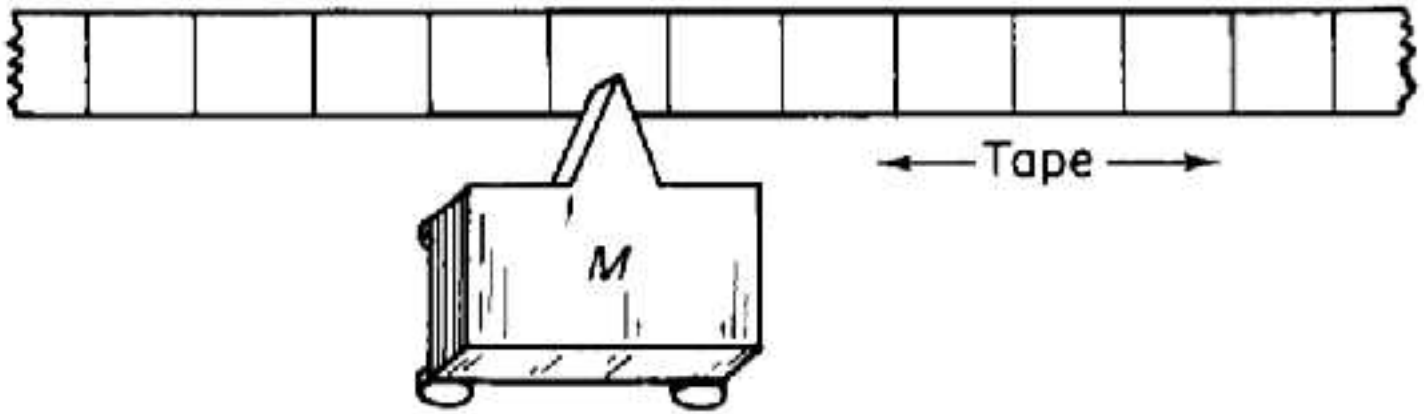
Less obvious things people do with the Turing machine

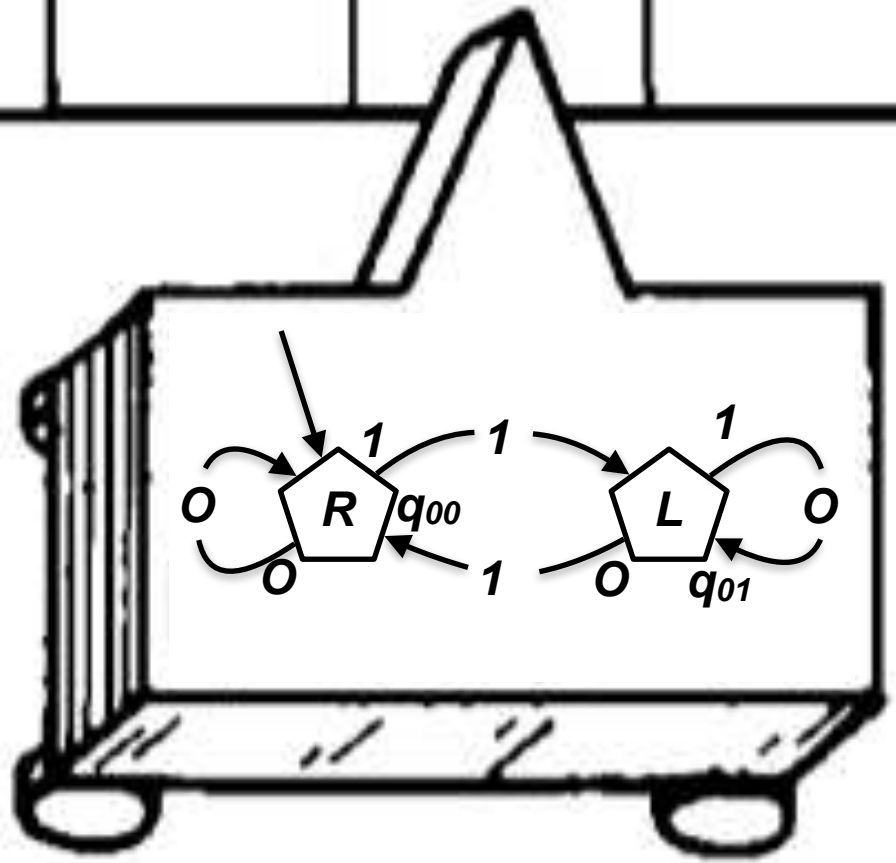
- Smallest Turing machine
- Busy Beaver record breaking
- Self-replicating Turing machines
- Turing machines in Lego
- Turing machines in wood
- Turing machines in Minecraft
- Turing machine tattoos
- Turing drawings
- https://youtu.be/soJ3FPvs7QI?t=227
- ...

Why would anyone want to hack a UTM?

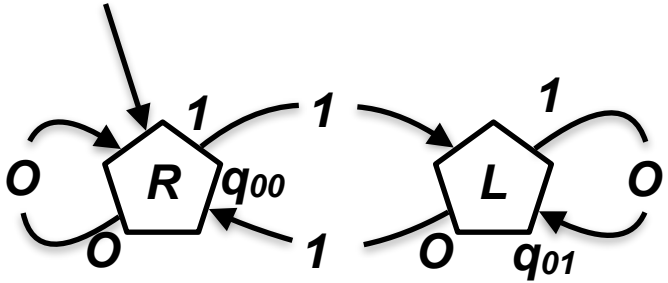
Why are computers so often insecure?

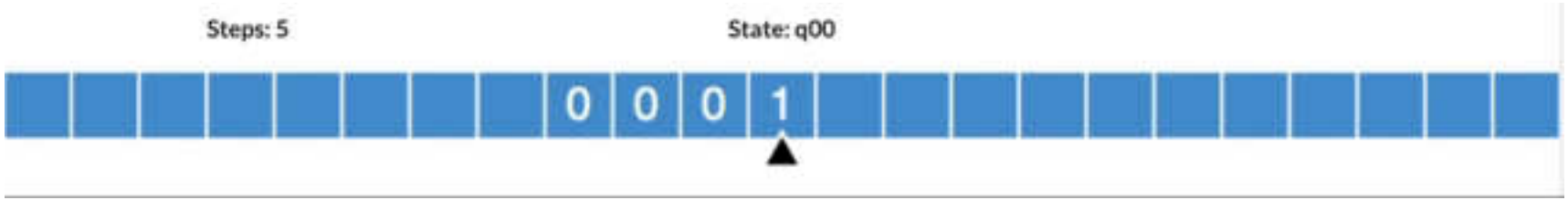
A Turing machine



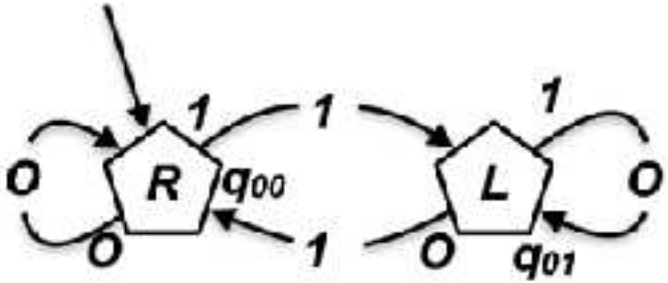

1
1
1
O
R
q00
L
O
O
1
O
q01

A binary counter Turing machine

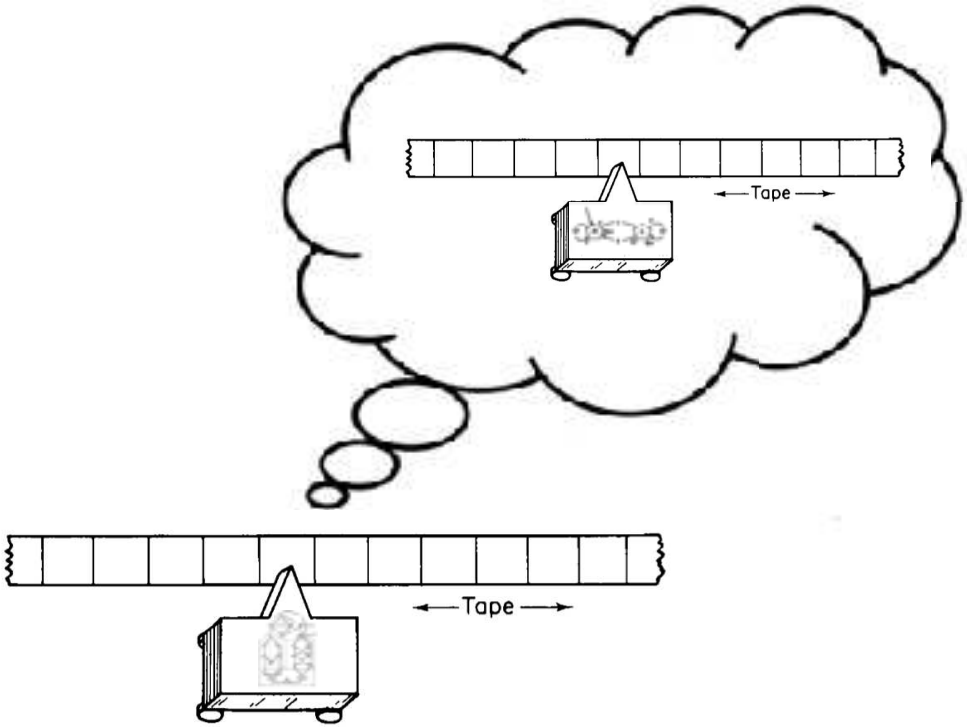




Input to a Turing machine

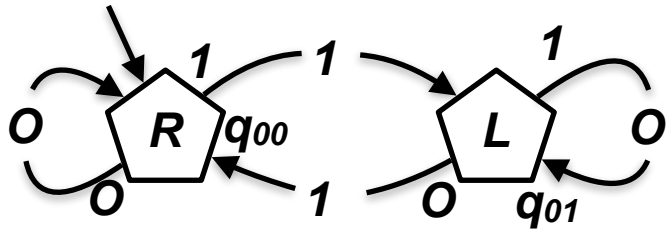


A Universal Turing Machine
A Turing machine that simulates Turing machines



Marvin Minsky's universal Turing machine

A binary counter Turing machine

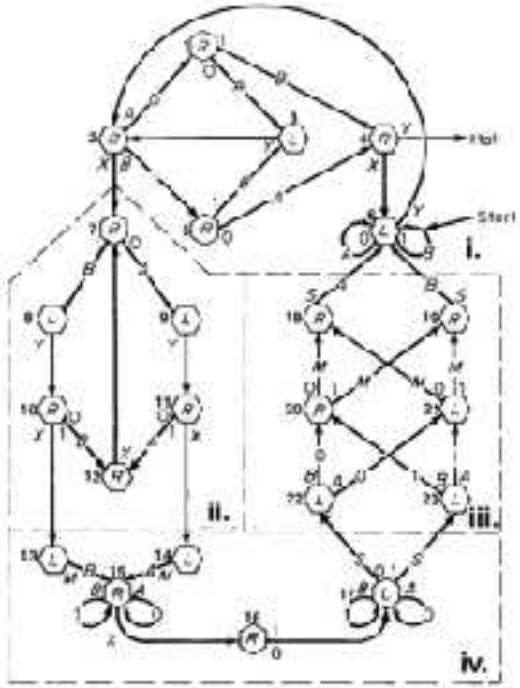


(old state, symbol scanned, new state, symbol written, direction of motion)

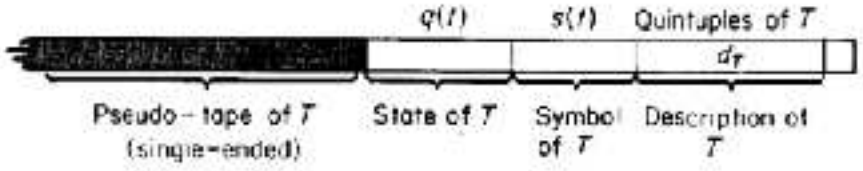
$(q_{00}, 0, q_{00}, 0, R)$ $(q_{01}, 0, q_{00}, 1, R)$
$(q_{00}, 1, q_{01}, 1, L)$ $(q_{01}, 1, q_{01}, 0, L)$

(00, 0, 00, 0, 1) (01, 0, 00, 1, 1)
(00, 1, 01, 1, 0) (01, 1, 01, 0, 0)

X0000001X0010110X0100011X0110100Y



...000M000 Y 00 1 X 000...



Simulating a binary counter

https://intrinsic-propensity.github.io

DIAGRAM CONVENTIONS

In most of our machines each state will have the character of an unidirectional search: each state is usually associated with moves in a single direction. The diagrams can be made simpler and more transparent by recognizing this fact. Thus we can represent the machines of 6.1.1 and 6.1.2 by the diagrams in Fig. 6.1-1. Each arrow in the diagram represents

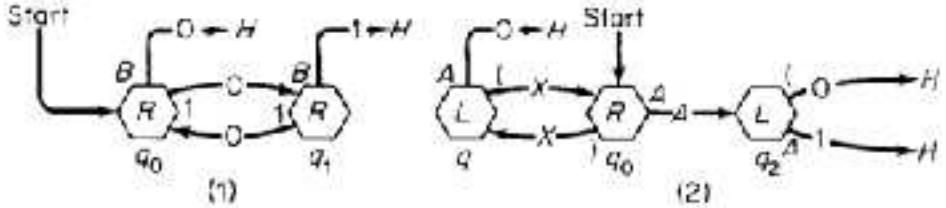


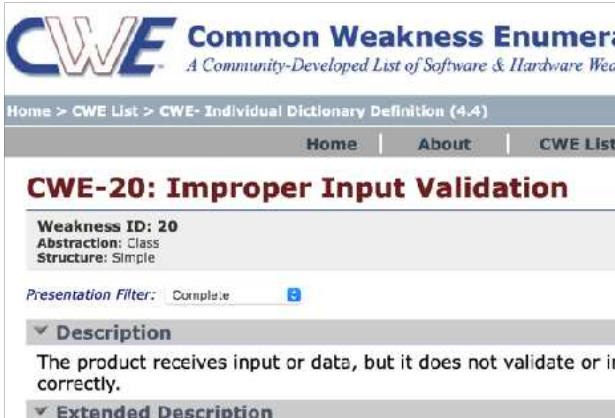
Fig. 6.1-1

some quintuple $(q_i, s_j, q_{ij}, s_{ij}, d_{ij})$. Then q_i is the state at the tail of the arrow, s_j is the symbol at its tail, s_{ij} is written in the middle of the arrow and omitted if the same as s_j, q_{ij} is the state at the head of the arrow, and d_{ij} is the symbol written inside the hexagon for q_{ij}. If two q_{ij}'s name the same state but their d_{ij}'s are different, we cannot use this kind of diagram.[1] The most common quintuples, of the form $(q_i, s_j, q_i, s_j, d_{ij})$ are simply omitted.

A vulnerability

Minsky's rule reduces complexity
Explicitly adding all implicit quintuples to the diagram would require close to 70 arrows in addition to the 45 explicitly drawn in the diagram.

However, Minsky's rule creates approximately twice as many implicit quintuples as the 70 required.



Improper input validation

https://intrinsic-propensity.github.io

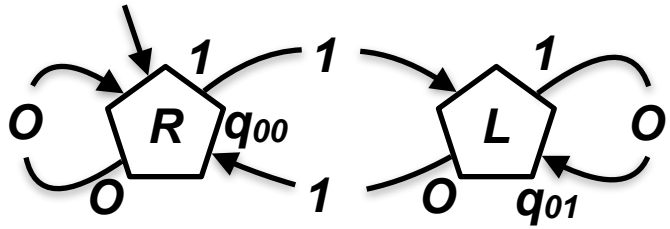
What does it even mean to hack an UTM?

Malicious SQL input to web page that allows arbitrary code execution

```
105; DROP TABLE Suppliers
```

Result

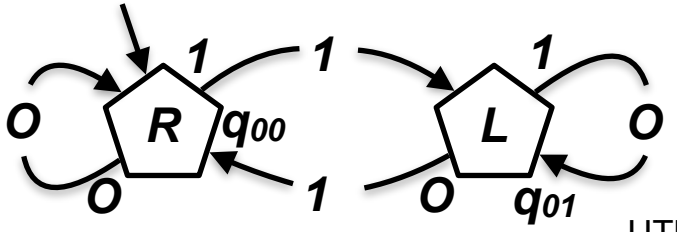
```
SELECT * FROM Users WHERE UserId = 105; DROP TABLE Suppliers;
```



Malicious input to simulated Turing machine that allows arbitrary code execution?

```
...00ΔΔ...ΔM000Y00ΔX0000001X0010110X0100011X0110100Y0...
```

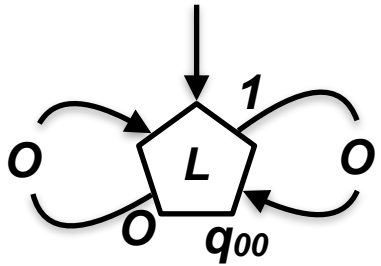
Start: Binary counter



UTM in State 6

```
...00ΔΔ...ΔM000Y001X0000001X0010110X0100011X0110100Y0...
```

Target: Eraser

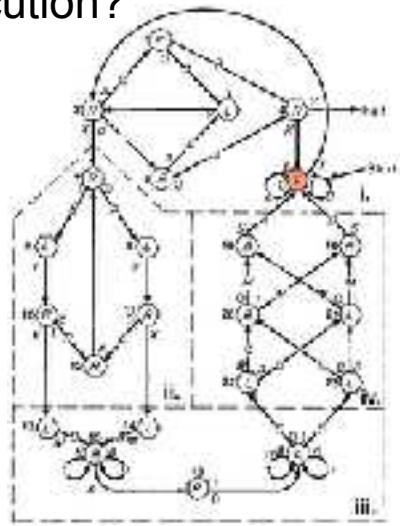


UTM in State 6

```
...00DD...DMY001X0000000X0010000Y0...
```

Hacker's objective

Is there any input ΔΔ...Δ that will coerce the UTM into a target tape, state and head position chosen by the attacker at some point in the execution?

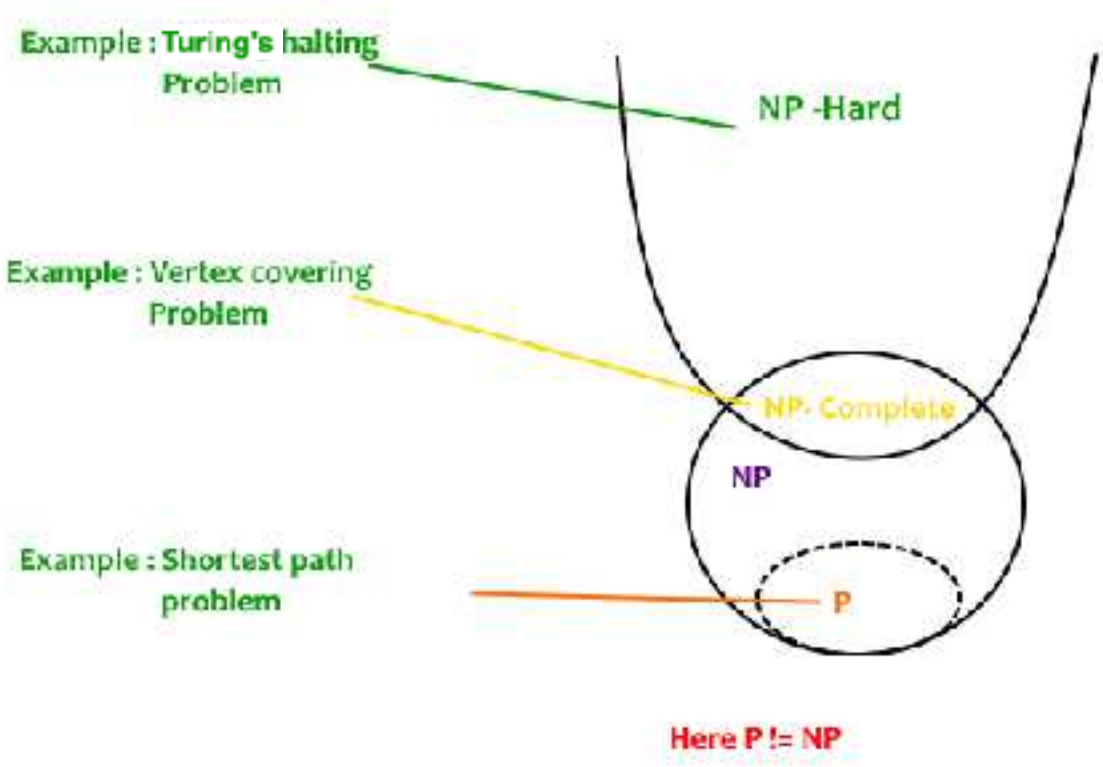


Is hacking in NP? P? NP-hard?

Brute-forcing will be difficult:

Number of possible strings is 8^n where n is the number of input symbols.

($8^{21} \approx 10^{19}$)



Inspiration from stack buffer overflow attacks

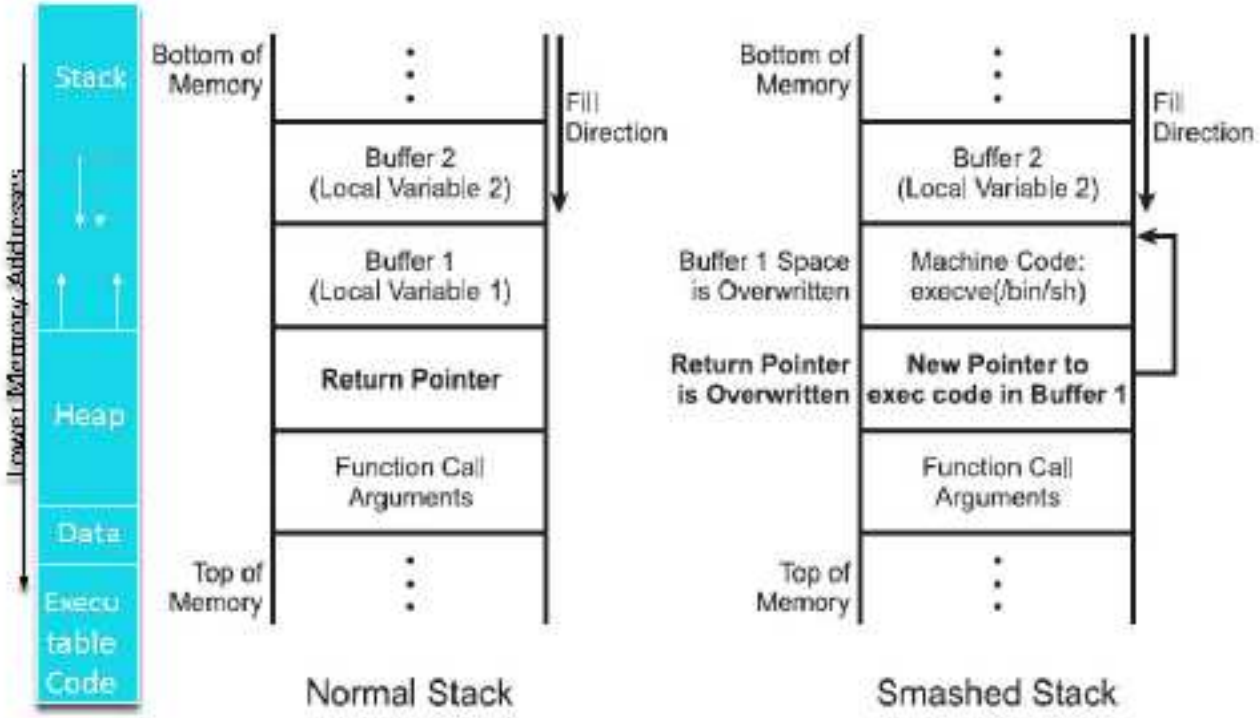


EXHIBIT 10.2 A normal stack and a stack with a buffer overflow.

Strategy

Origin

...00ΔΔ...ΔM000Y001X0000001X0010110X0100011X0110100Y0...

Target

...00DD...DM000Y001X0000000X0010000Y0...

Strategy

Origin

...00ΔΔ...ΔM000Y001X0000001X0010110X0100011X0110100Y0...

Target

...00DD...DM000Y001X0000000X0010000Y0...

Naïve exploit attempt

...00DD...DY000X0000000X0010000Y0M000Y001X0000001X0010110X0100011X0110100Y0...

Naïve exploit attempt

https://intrinsic-propensity.github.io

Strategy

Origin

...00ΔΔ...ΔM000Y001X0000001X0010110X0100011X0110100Y0...

Target

...00DD...DM000Y001X0000000X0010000Y0...

Naïve exploit attempt

...00DD...DY000X0000000X0010000Y0M000Y001X0000001X0010110X0100011X0110100Y0...

Working exploit

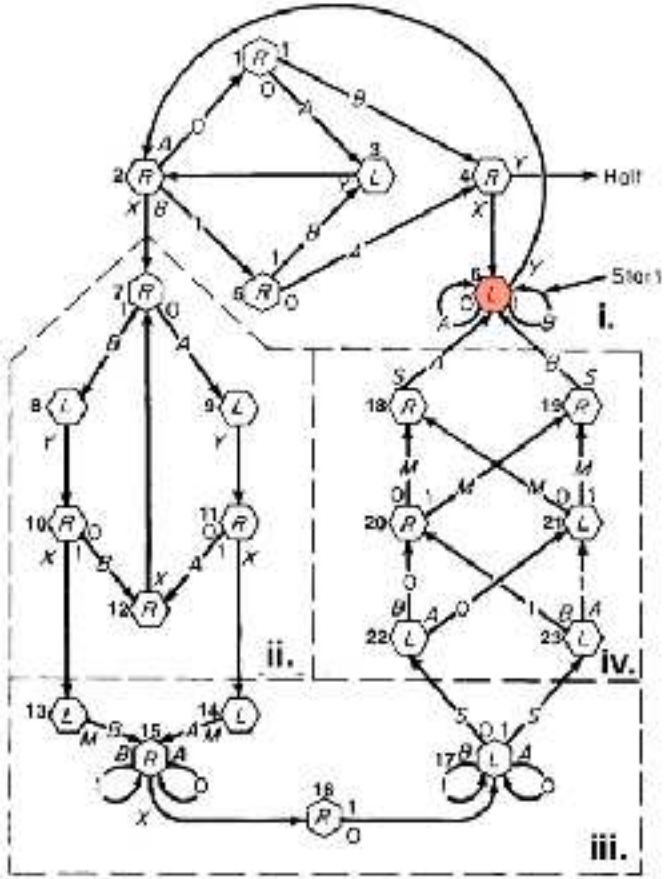
...00DD...DYBAAXAAAAAAAAXAABAAAASM000Y001X0000001X0010110X0100011X0110100Y0...

Arbitrary code execution

https://intrinsic-propensity.github.io

Mitigations

- Validate inputs by preprocessing
- Specify all required quintuples (an additional 70) explicitly
- Use a special alphabet for *T*'s tape
- Find (rare) unexploitable parameters (e.g. buffer between M and Y <3)



...00ΔΔ...ΔM000Y001X0000001X0010110X0100011X0110100Y0...

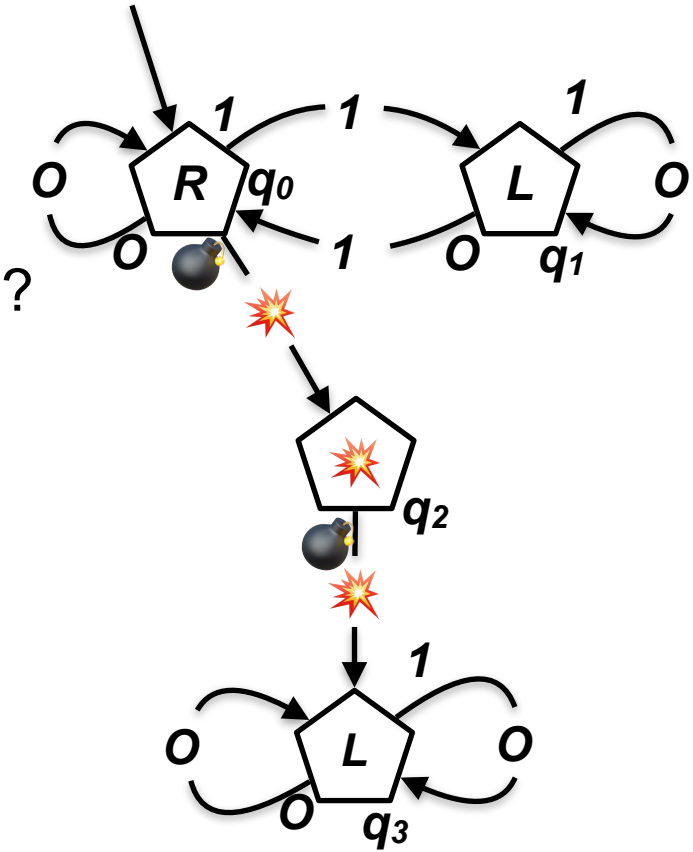
Why are computers so often insecure?

Why is even an implementation of the simplest imaginable computer vulnerable to arbitrary code execution?

Is it the case that computers are intrinsically brittle – that they at their very core have a propensity for arbitrary code execution vulnerabilities?

Insecurity root cause hypotheses

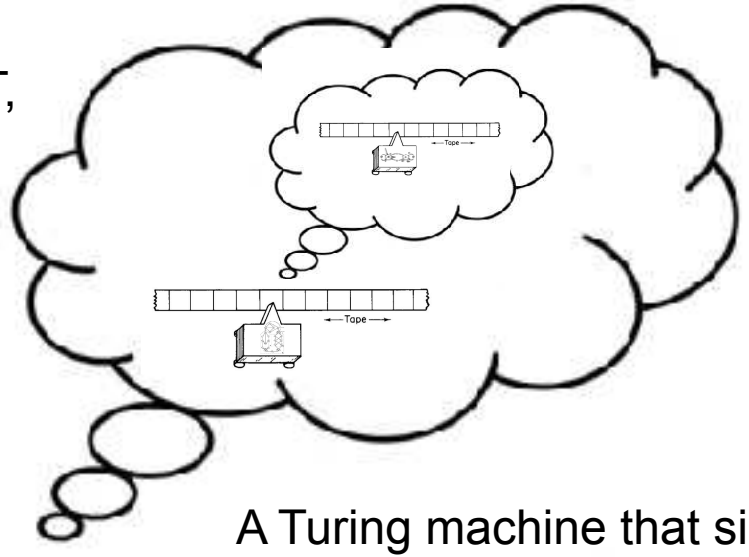
- Complexity?
- Human fallibility?
- John von Neumann's stored program?
- Weird machines?
- ...



Dullien, Thomas. "Weird machines, exploitability, and provable unexploitability." *IEEE Transactions on Emerging Topics in Computing* 8.2 (2017).

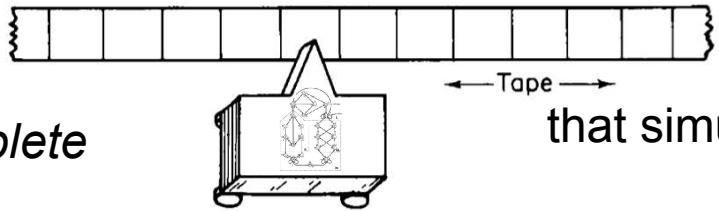
Another perspective on arbitrary code execution hacking

The discovery that a Turing machine T, simulated on a universal Turing machine U, in itself is a universal Turing machine



A Turing machine that simulates a Turing machine that simulates Turing machines

The discovery that T is *accidentally Turing complete*



Many things are accidentally Turing complete

- Lego
- MineCraft
- x86 `mov`
- Musical notation
- Magic: The Gathering
- The smallest Turing machine

Concluding hypothesis

Computers are intrinsically brittle – at their very core, they have a propensity for arbitrary code execution vulnerabilities – because rule-based systems have a propensity to Turing completeness