The Science of Insecurity

Len Sassaman
Meredith L. Patterson
Sergey Bratus

Tribute to Len Sassaman

- Anonymity/privacy researcher, cypherpunk
- Moved to language-theoretic security in 2009
- Because the future of an open Internet depends on smoothing out the attack surface

1980 - 2011

Insecurity is the “new normal”

- “Treat all systems as compromised”
  - “There's no such thing as ‘secure’ any more.” -- Deborah Plunkett, NSA Information Assurance Directorate
- “Long weeks to short months before a security meltdown” – Brian Snow, in December 2010
  - “are we there yet?” You bet we are, unless one agrees to view LulzSec as “APT”/nation state

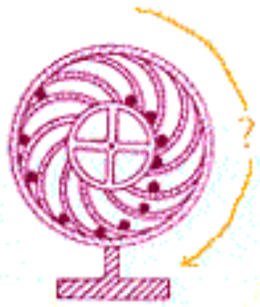
Not for lack of trying

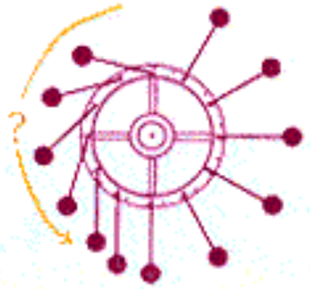
- Various “trustworthy computing” initiatives
- Lots of “secure coding” books
- Mounds of academic publications
- New hacker-developed testing methods: fuzzing, RE-backed binary analysis …

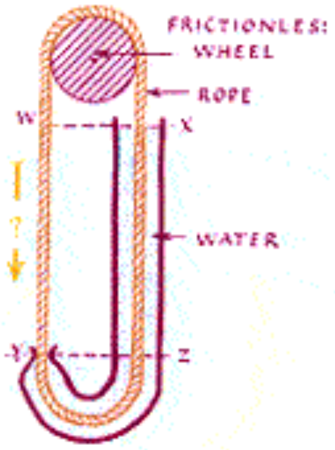
- Yet software still sucks!
- And hardware – we don’t even know how much it sucks (no tools to poke its attack surface -- yet)

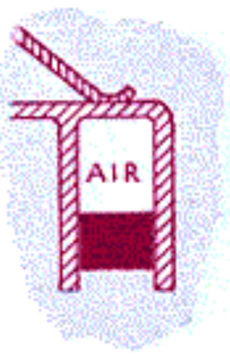
There must be something we are doing wrong

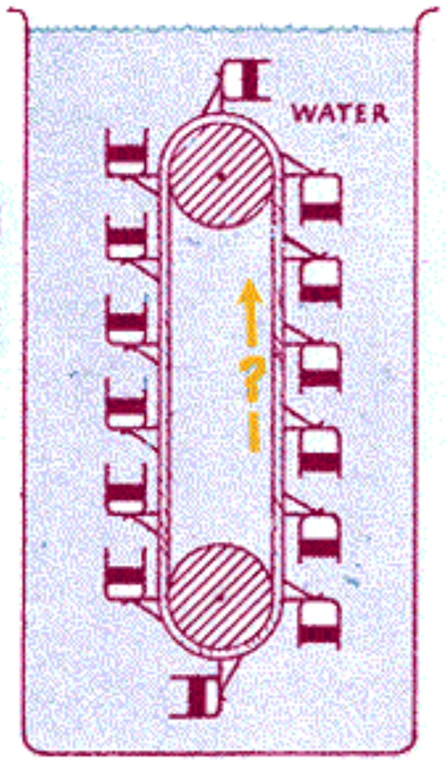
- Science to engineers: some problems are **not solvable**, do not set yourself up to solve them











“There is a **law of nature** that makes it so, no matter how hard you try”

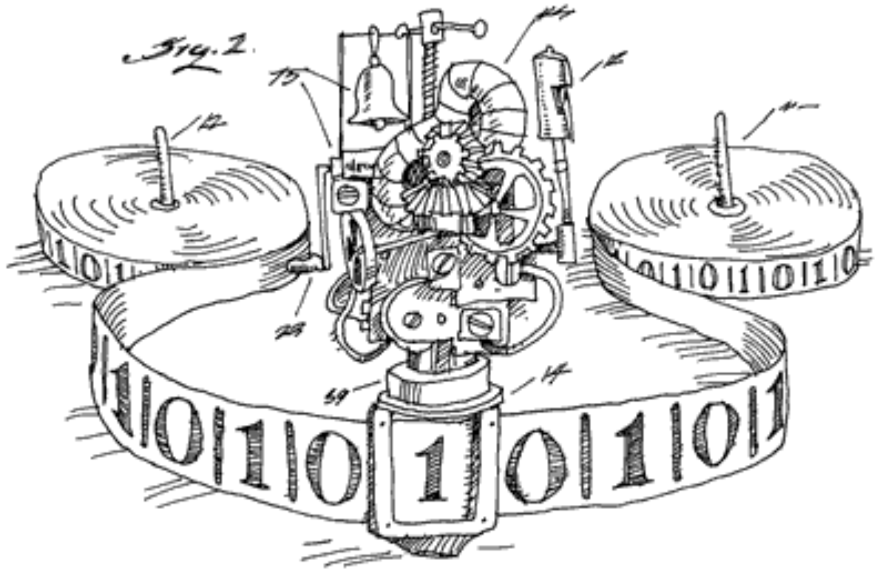
(In)security is all about computation

- **(In)security** is inseparable from **computation**: *trustworthiness* of a computer system is **what the system can and cannot compute**
  - Can the system decide if an input is invalid/unexpected/malicious & reject it?
  - If target has no “shell” and no place to run one, it can be trusted to not drop a shell
  - Will program perform only expected computations, or “malicious” ones, too? ***Etc.***

Turing machines and undecidability

*Turing Machine: t*he model of computer to study the limits to what is ***computable***

TM can do what your computer, phone, keyboard, NIC, … can do

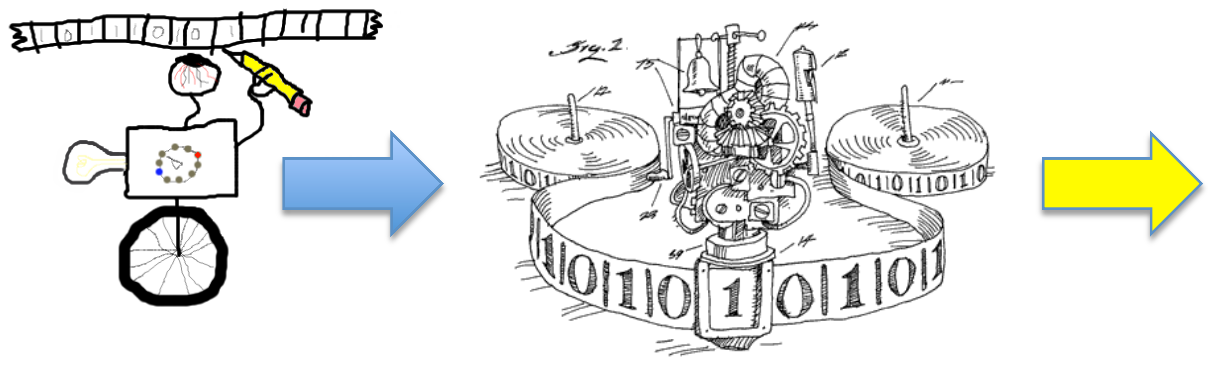


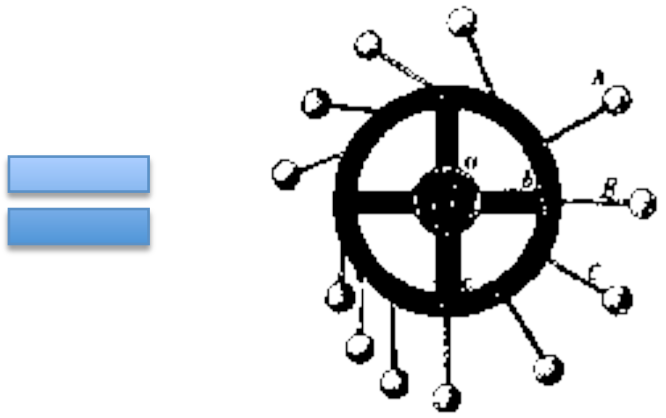
Undecidable problems: No TM can solve them.

"The Halting Problem is Undecidable"

Cornerstone: the Halting Problem

- "I can build a TM that takes another TM as input and decides if it will terminate"





The history of the Uncomputable

- Leibniz: "*Can a machine determine truth values of mathematical statements*"? [17th century]
- Hilbert's Entscheidungsproblem, [1928]
  - "*Is an arbitrary logical statement valid or invalid*"?
- Church [1936], Turing [1937]: ***Negative!***
  - Based on work by Kleene, Goedel [1930s]
- Curry-Howard correspondence: programs are proofs and vice versa

Programs and exploits as proofs

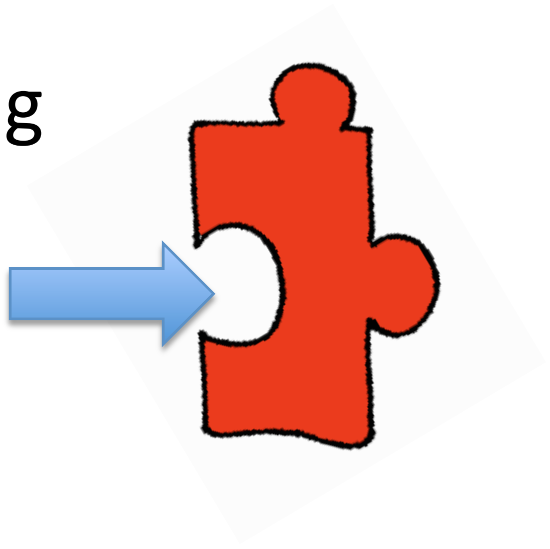
- **Curry-Howard** correspondence: **programs** are **proofs** and vice versa
- **Exploits** are proofs too: by construction of unexpected/hostile computation
- Formal Duality? <TBD>

Is insecurity due to program/protocol designs setting us up to solve **undecidable** problems? **YES!**

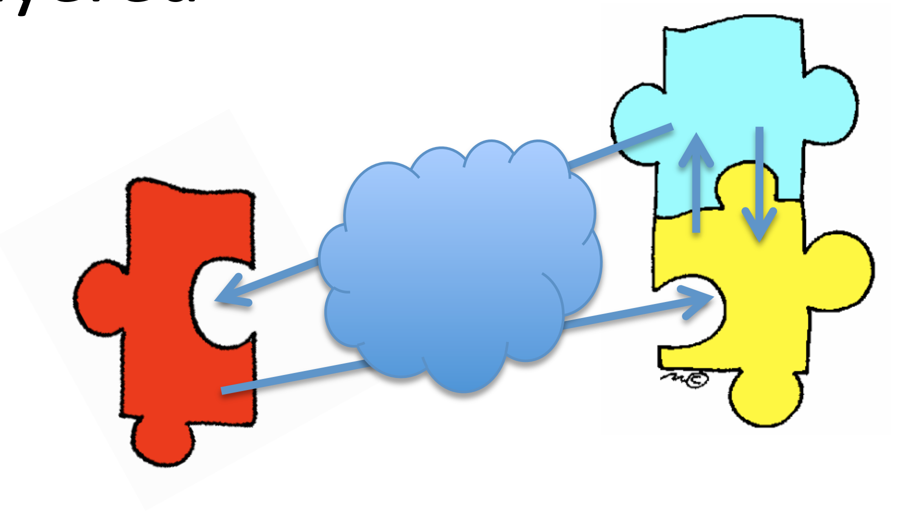
- Some designs force programmers to "solve" the **undecidable** Halting Problem w.r.t. inputs and communications.
  - Bad news: no amount of testing or "fixing" helps
  - No real need for that, either
- How did we get there and how to fix it?

Basic requirements in a composed world

- One component/program accepting inputs
  - Must accept or reject messages safely



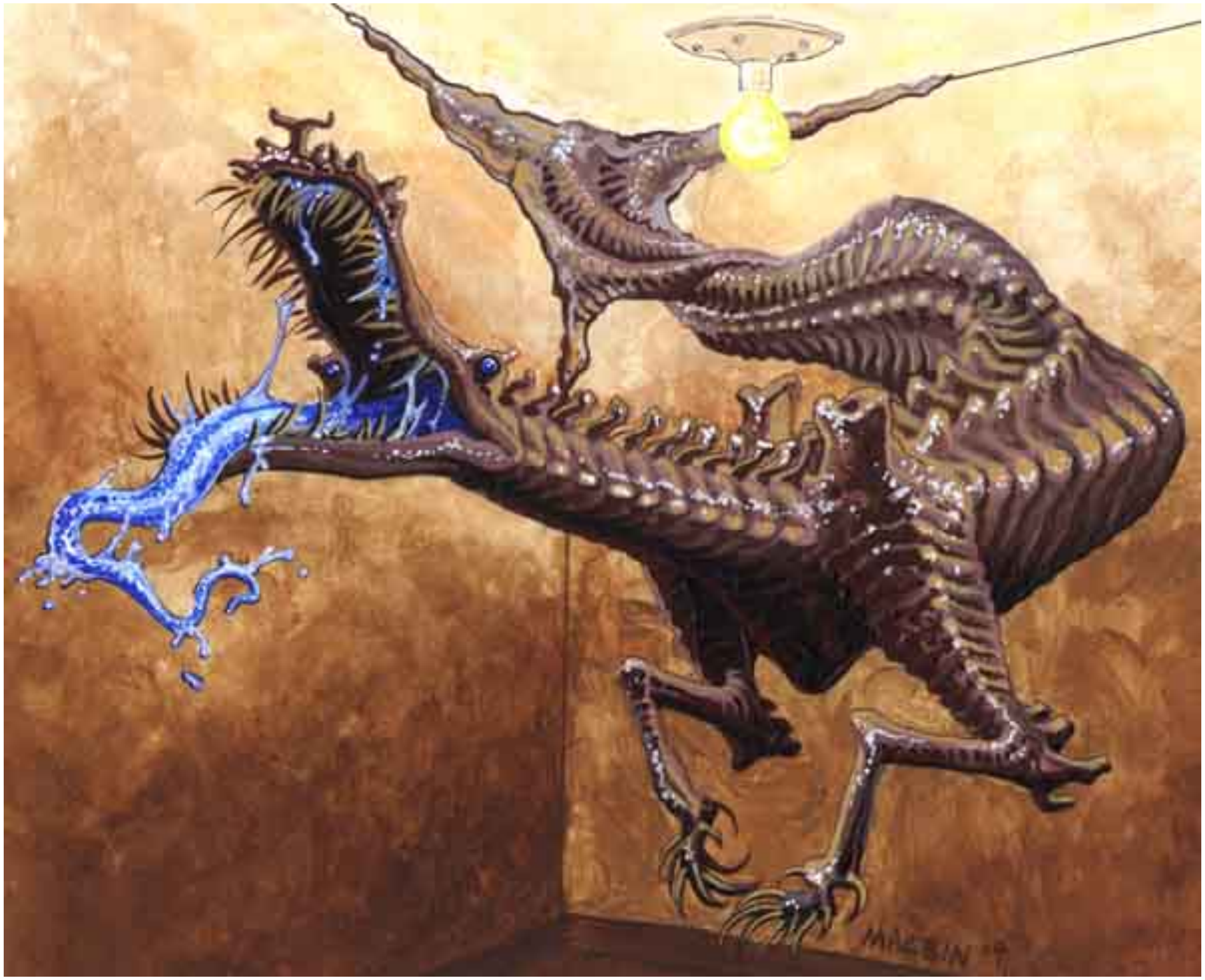
- Components communicating (distributed system, layered architectures)
  - Messages must be interpreted identically by endpoints



The Hidden Recognizer

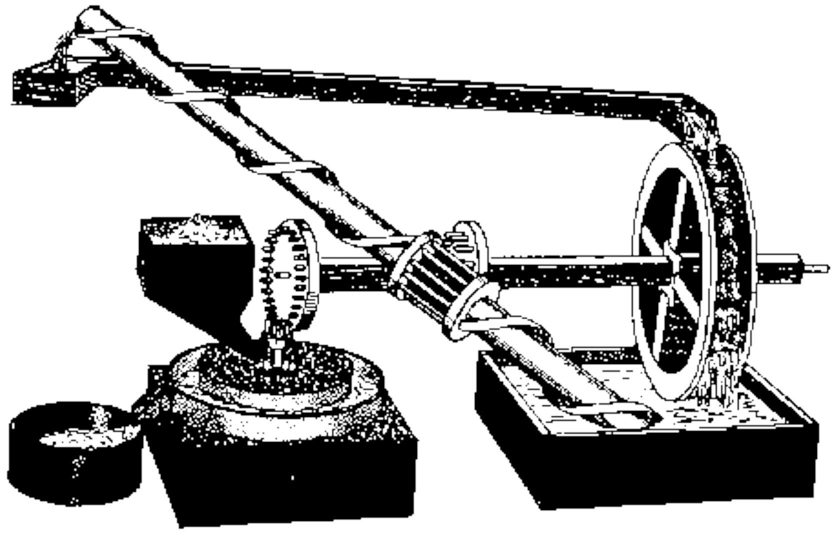
- Both of the above are **language recognition problems,** so Turing/Church MUST be brought to bear to settle/study them
- The deadliest programming pattern: "**The Scattered Recognizer**"
  - Checks for input validity are scattered throughout the program, mixed with processing logic
- Implicitly/vaguely understood input languages: the mother of all Halting Problems
  - We'll show exactly how the Turing Beast arises

The **Turing Beast** Computation



No 80/20 for the Halting Problem

- There is no “80/20” engineering solution to the Halting Problem -- or Perpetual Motion
  - No amount of **testing** or “**fixing**” will help
  - If someone is selling it, run away
  - You don’t want to be around when it breaks



What is INsecurity?

- Holes for sneaking in executable code?
  - "Malicious code" has been deprecated for "malicious computation" (since 2000)
- Memory corruption?
- In-band signaling?
- Exposing unnecessary privilege?
- All of the above?

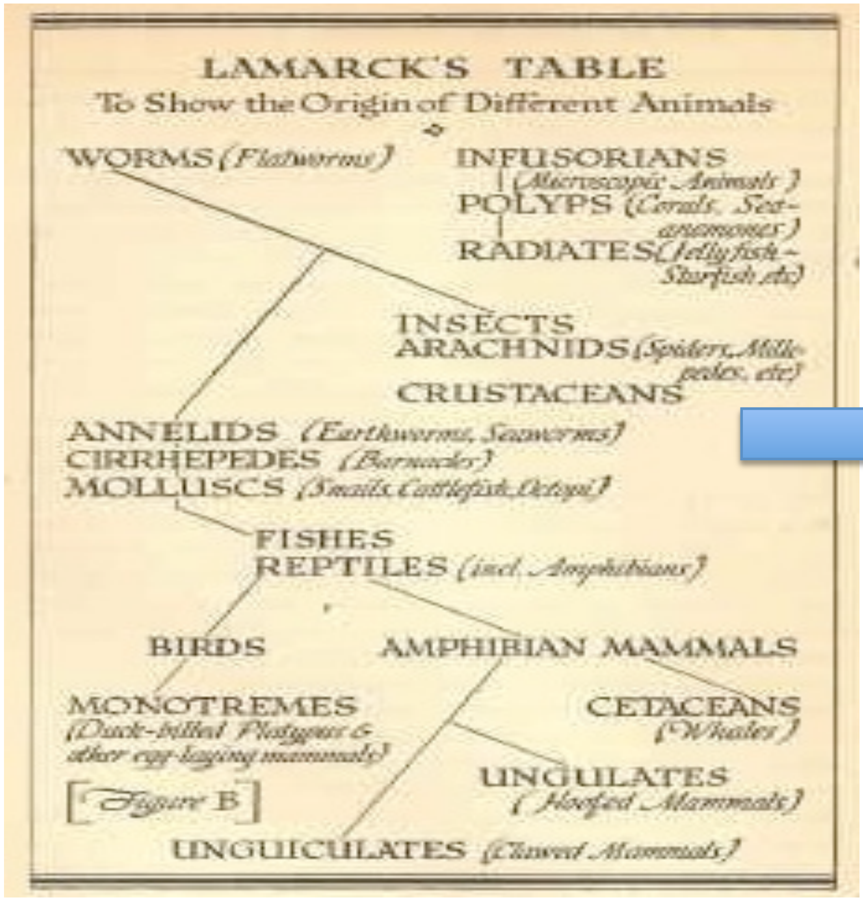
“Top N” vulnerability classifications?

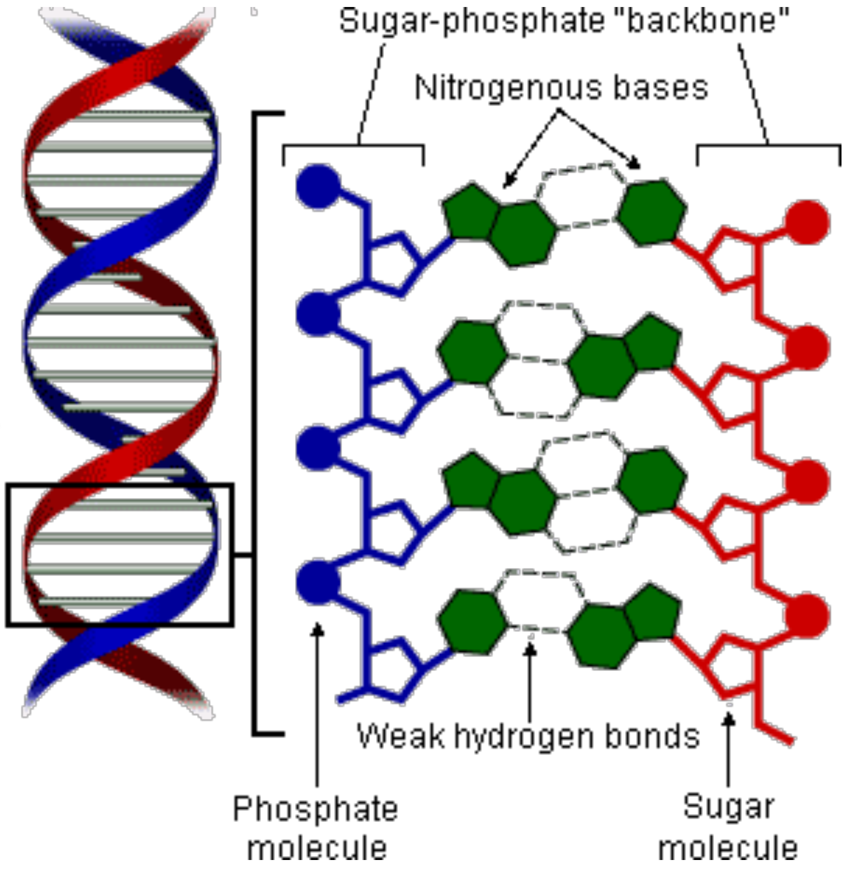
[In] a certain Chinese encyclopaedia entitled *‘Celestial Empire of benevolent Knowledge’* … the animals are divided into: (a) belonging to the emperor, (b) embalmed, (c) tame, (d) sucking pigs, (e) sirens, (f) fabulous, (g) stray dogs, (h) included in the present classification, (i) frenzied, (j) innumerable, (k) drawn with a very fine camelhair brush, (l) et cetera, (m) having just broken the water pitcher, (n) that from a long way off look like flies.

--- *Jorge Luis Borges, “The Analytical Language of John Wilkins”*

Nature and origins of insecurity:

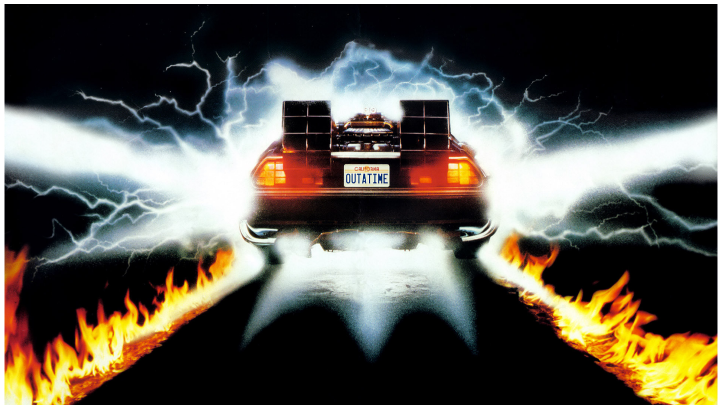
Need a leap from “Lamarck” to “Watson and Crick”


LAMARCK'S TABLE
To Show the Origin of Different Animals
WORMS (Flatworms)
INFUSORIANS (Microscopic Animals)
POLYPS (Corals, Sea-anemones)
RADIATES (Jellyfish-Starfish etc)
INSECTS
ARACHNIDS (Spiders, Millepedes, etc)
CRUSTACEANS
ANNELIDS (Earthworms, Seaworms)
CIRRHEPEDES (Barnacles)
MOLLUSCS (Snails, Cuttlefish, Octopi)
FISHES
REPTILES (incl. Amphibians)
BIRDS
AMPHIBIAN MAMMALS
MONOTREMES (Duck-billed Platypus & other egg-laying mammals)
CETACEANS (Whales)
UNGULATES (Hoofed Mammals)
UNGUICULATES (Clawed Mammals)
[Figure B]



Back to the Turing Future!

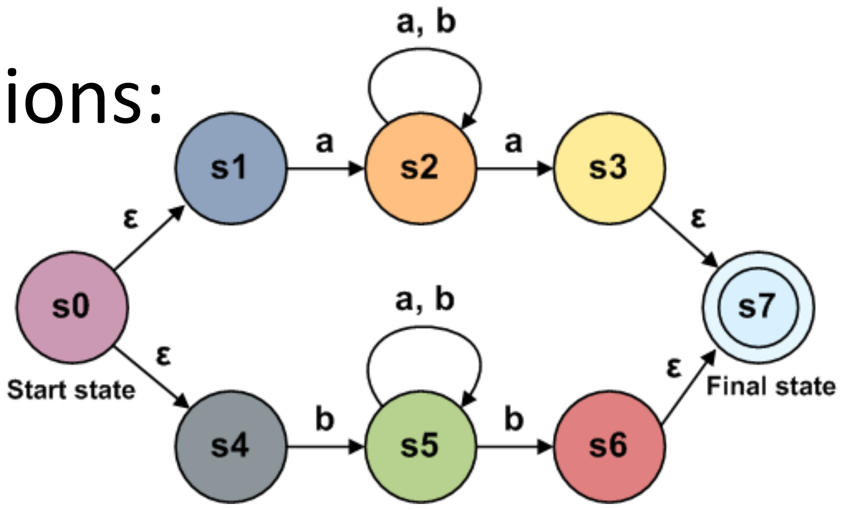
- Insecurity as related to computation must be understood from the **Turing** and **Church** basics of computation
  - Academics study computation models
  - Hackers study actual computational limits of **real systems**



The right machine for a task (“mostly harmless”)

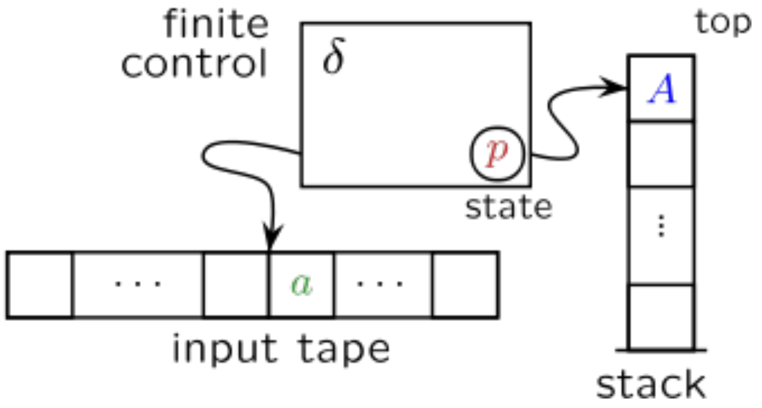
- Matching regular expressions: **finite state machines**

  a[ab]+a | b[ab]+b



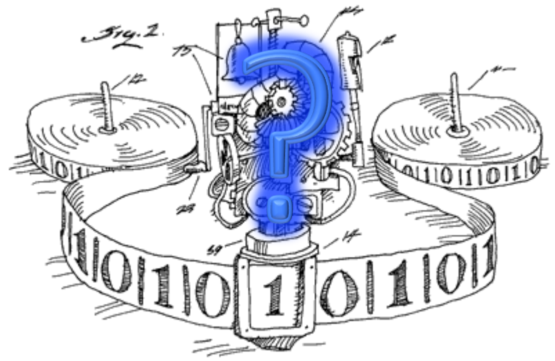
- Matching recursively nested structures: **pushdown automata**

  ((([[({(…)})]])))



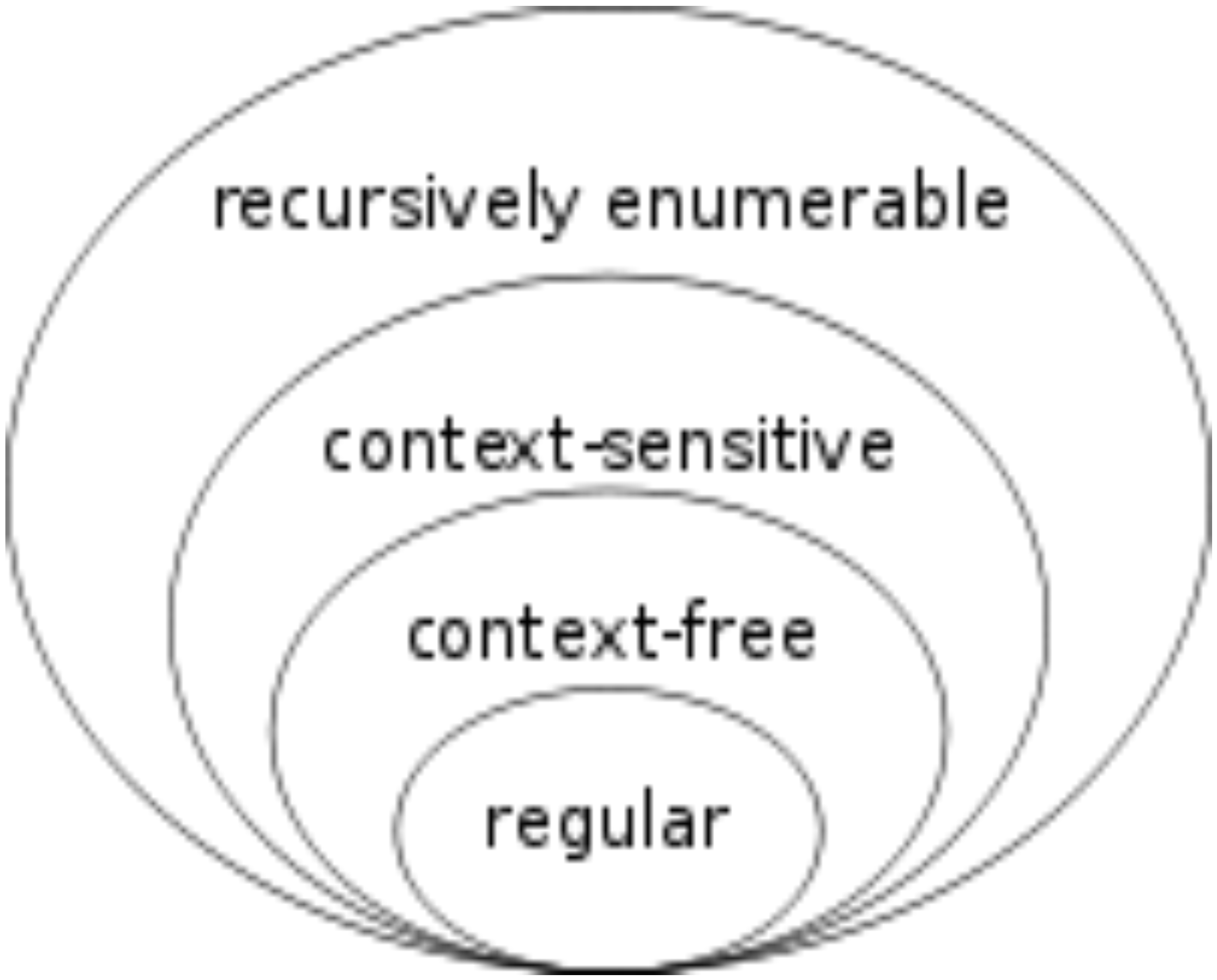
The right machine for a task (2)

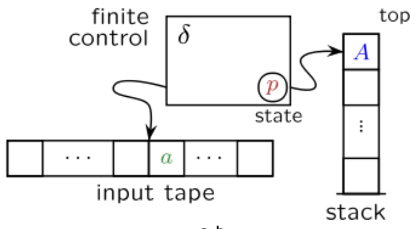
- Telling data from metadata ("weakly context-sensitive" protocols, e.g., those with **length fields)**
  - Parsing an IP packet past a few corrupted bytes

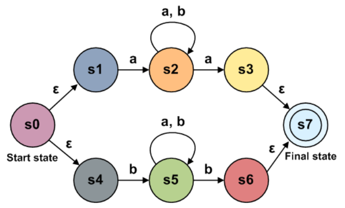


- Telling if input is a program that produces a given result: **UNDECIDABLE** (Rice's Theorem)

The language hierarchy

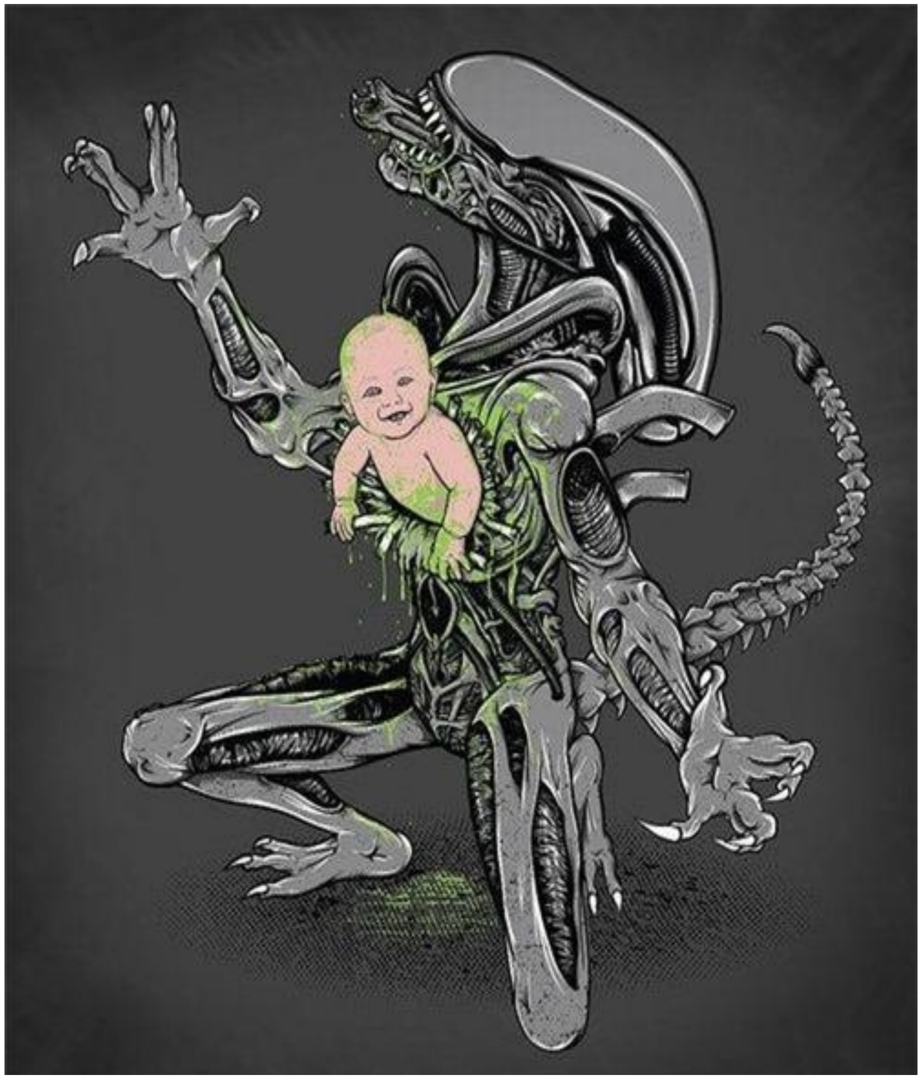






“Type I”: Input handling

In-component, “The weird machine inside”



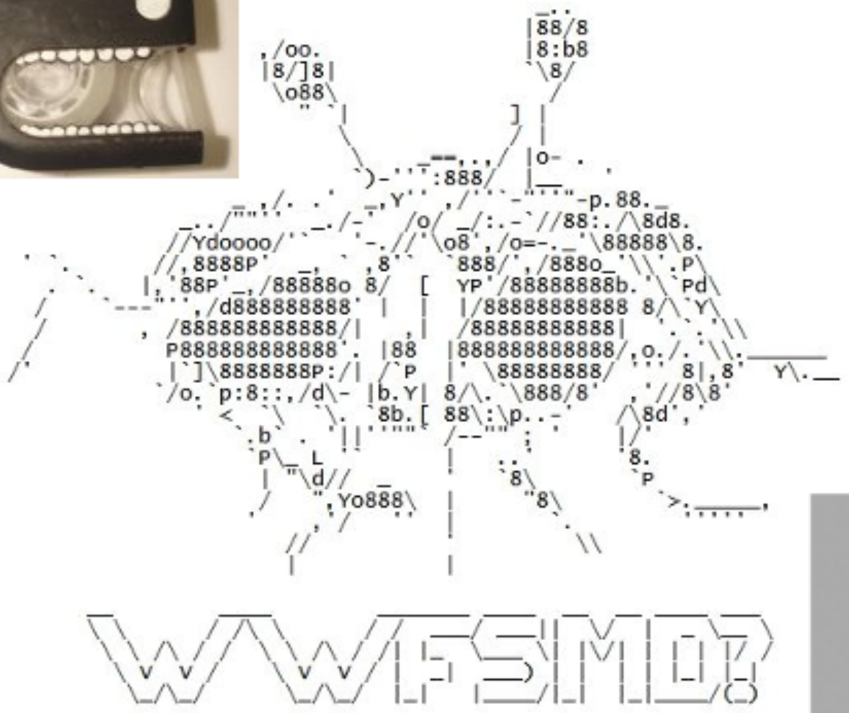
Insecurity: “What is exploitation?”

- Exploitation is unexpected computation

```
1119  sendmail: accepting connections
31337  \_ -bash
31338      \_ rm –rf /
```

- Exploits are **programs written in crafted inputs** that cause unexpected computation in the target system
- Programs for **what machines?** See Halvar Flake/Thomas Dullien talk @ Infiltrate 2011

“Weird Machines”



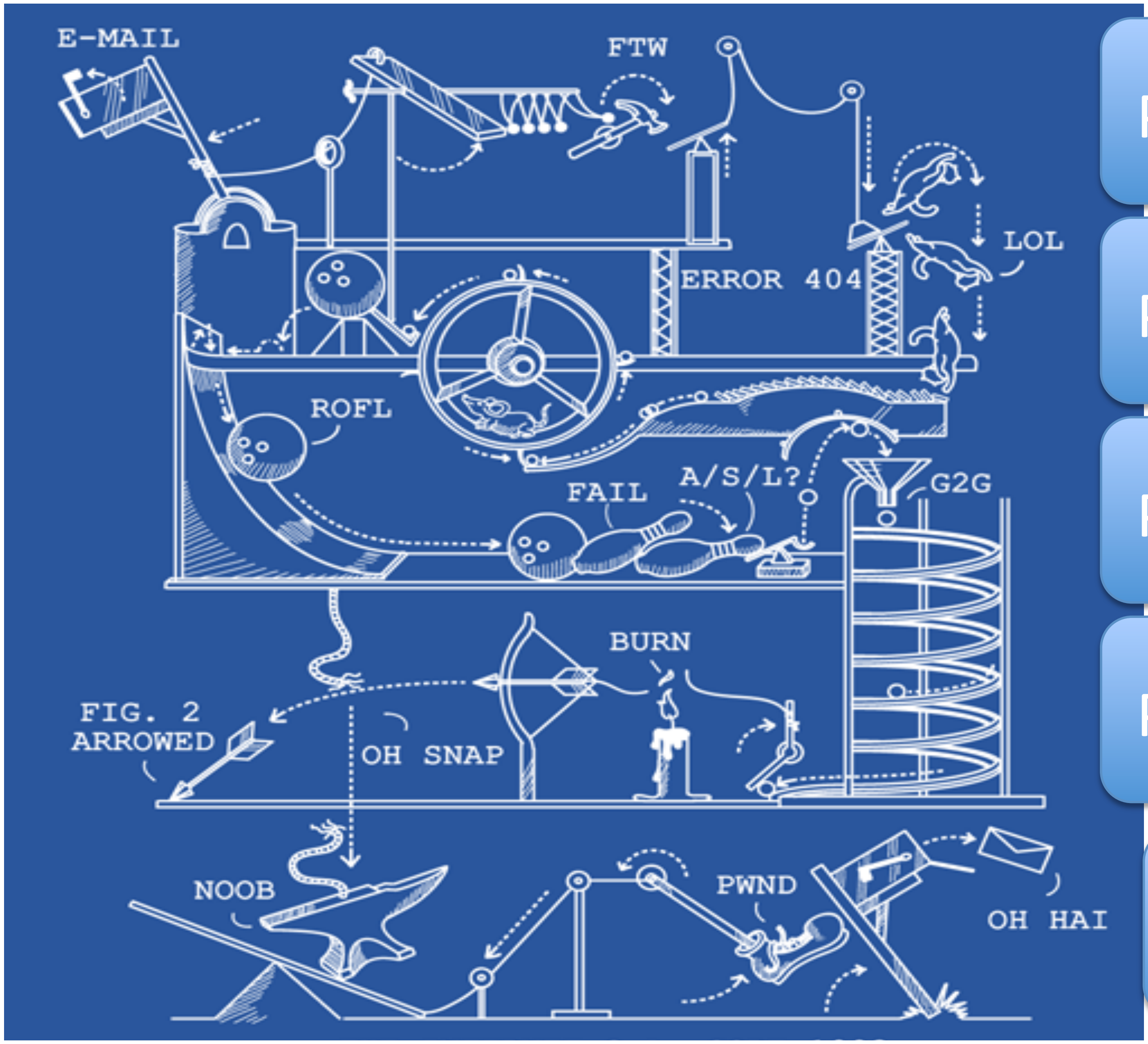
"Exploitation is setting up, instantiating, and programming a ***weird machine*** *"*

- A part of the target is overwhelmed by crafted input and enters an **unexpected** but **manipulable** state
- A "weird machine" is unleashed
  - Who let the dogs out?
- **Inputs** drive unexpected computation

Inside every program are weird machines waiting to break out

- Wherever **input** is accepted and processed by **ad-hoc code**, bugs are likely
- "***Deviant inputs***" cause memory corruption, type misinterpretation (e.g., signed vs unsigned integer overflow), etc.
- "**Primitive**" corrupting inputs are combined into full exploit-programs
  - See **gera**, **"*About Exploits Writing*"**, 2002
  - Our upcoming article in ;**login:** December 2011

“Bugs to primitives to exploits”



Phrack 57:8

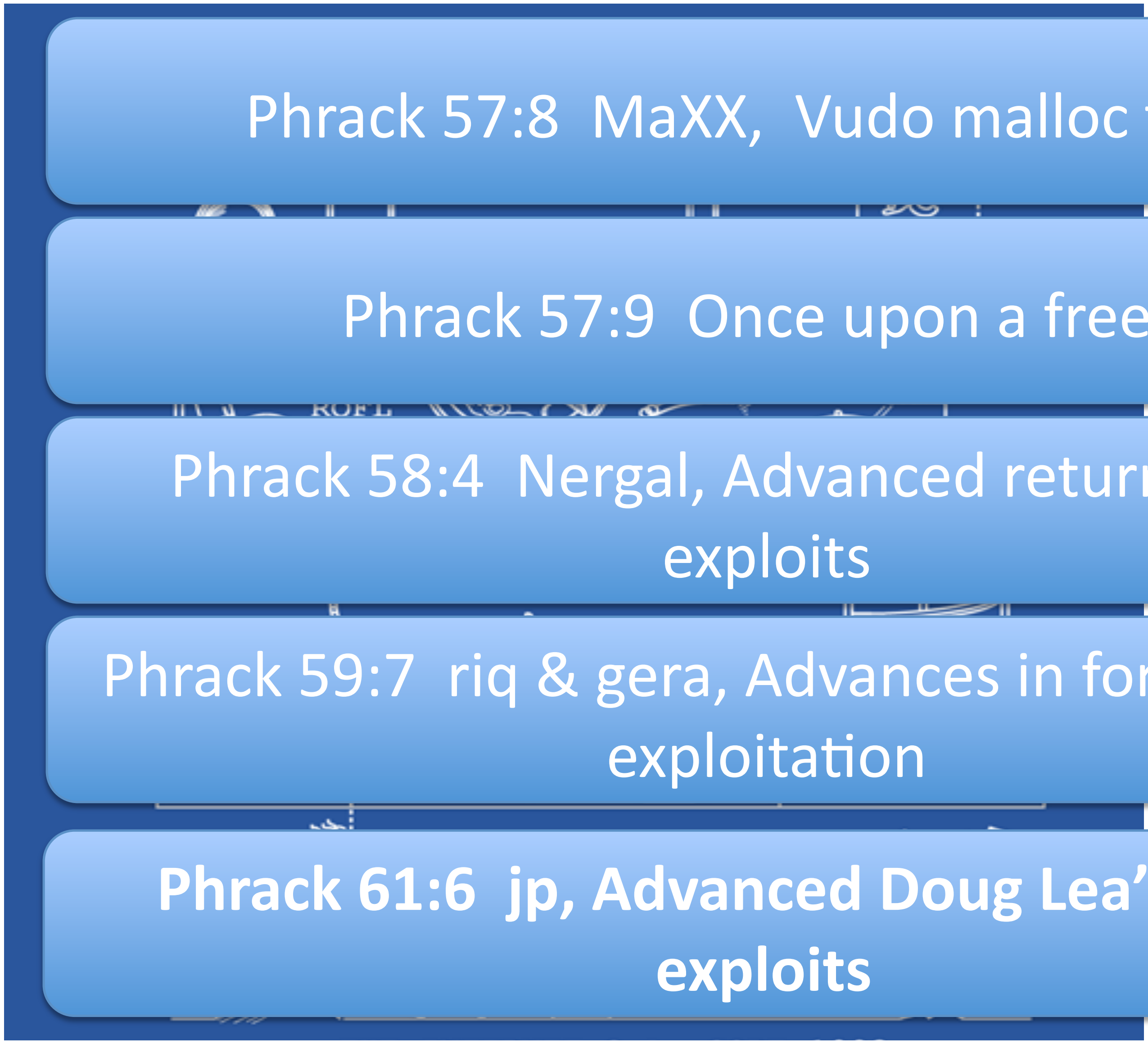
Phrack 57:9

Phrack 58:4

Phrack 59:7

Phrack 61:6

“Bugs to primitives to exploits”



Phrack 57:8 MaXX, Vudo malloc tricks

Phrack 57:9 Once upon a free()

Phrack 58:4 Nergal, Advanced return-to-libc exploits

Phrack 59:7 riq & gera, Advances in format string exploitation

Phrack 61:6 jp, Advanced Doug Lea’s malloc exploits

Occupy Input Handlers!

Is it **all** about parser bugs?

- No, but that’s a large chunk of it
- Every program component that receives input from others **has a recognizer for the language of valid or expected inputs**
  - Network stacks ~ frames/packets, protocols
  - Servers ~ valid requests (e.g. SQL injection)
  - Memory managers ~ valid heaps (metadata)
  - Function call flow ~ valid stacks

  …

An **implicit** recognizer is a **bad** recognizer

- **Ad-hoc** recognizer logic scattered throughout the program is **hard** to **test** and **debug**

- Lots of intermixed recognition/processing state => lots of unexpected states
  - "Weird machines run on borrowed state"
  - "Could you spare some state for my exploit?"

- Don't process what you cannot first recognize!

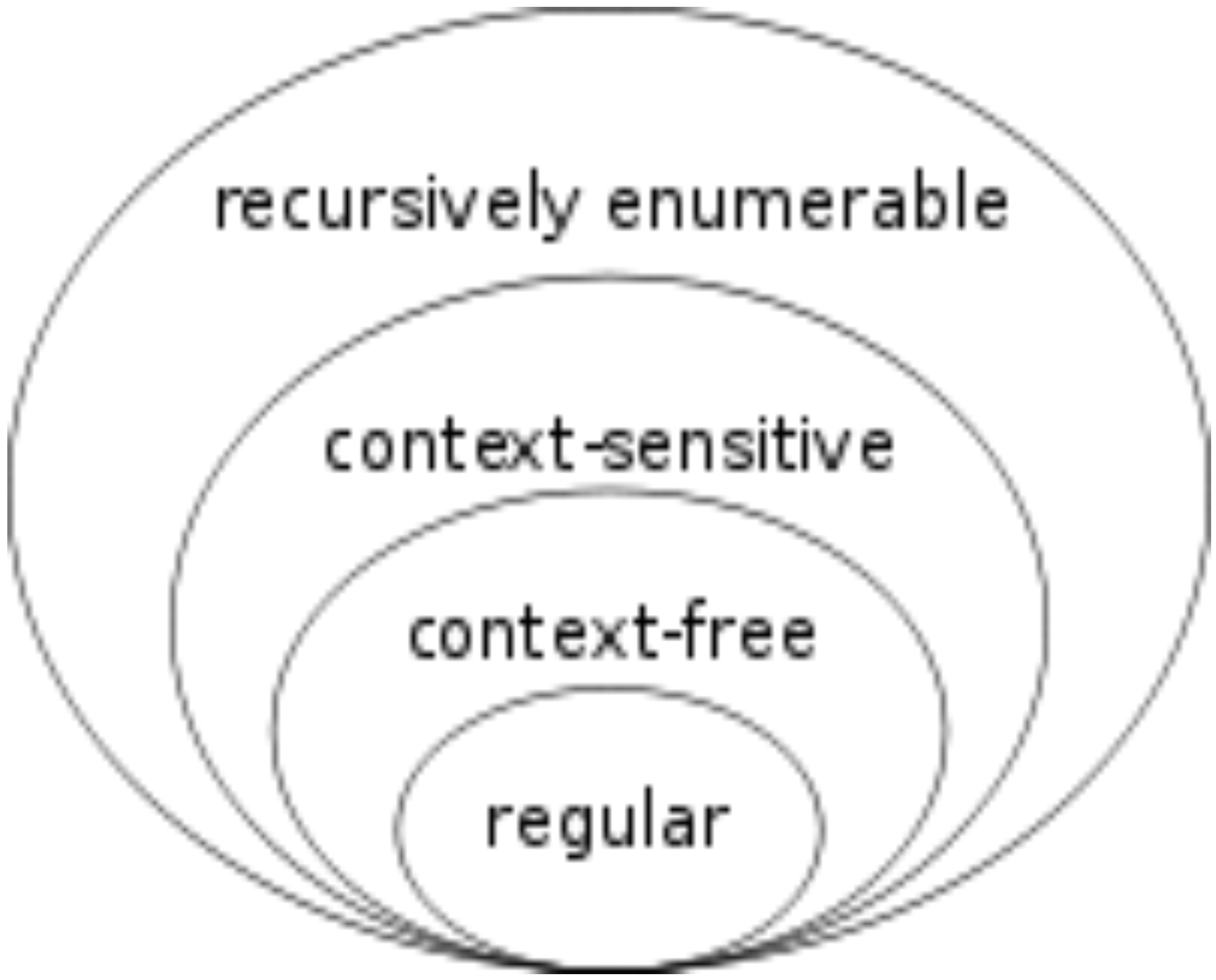
Occupy Program State!

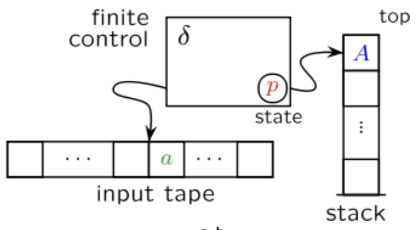
Regard all valid/expected inputs as a formal language

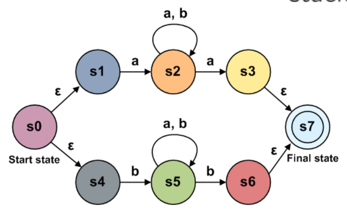
- **Know** and **be strict about** what your input language is
- Know what computational power it requires to get recognized
  - **Never** parse **nested structures** with **regexps**!
- Write the recognizer explicitly or, better, **generate** it from a **grammar**
- Stay away from Turing-complete input languages

Occupy Message Formats!

"Regular is a safe place to be"





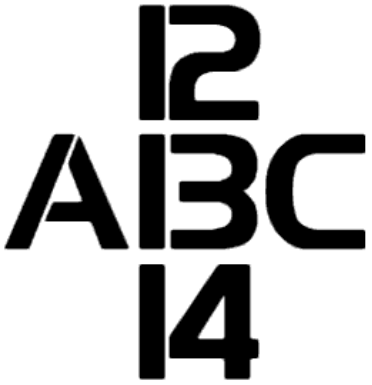


Occupy Protocol Design!

II. Composition & communication

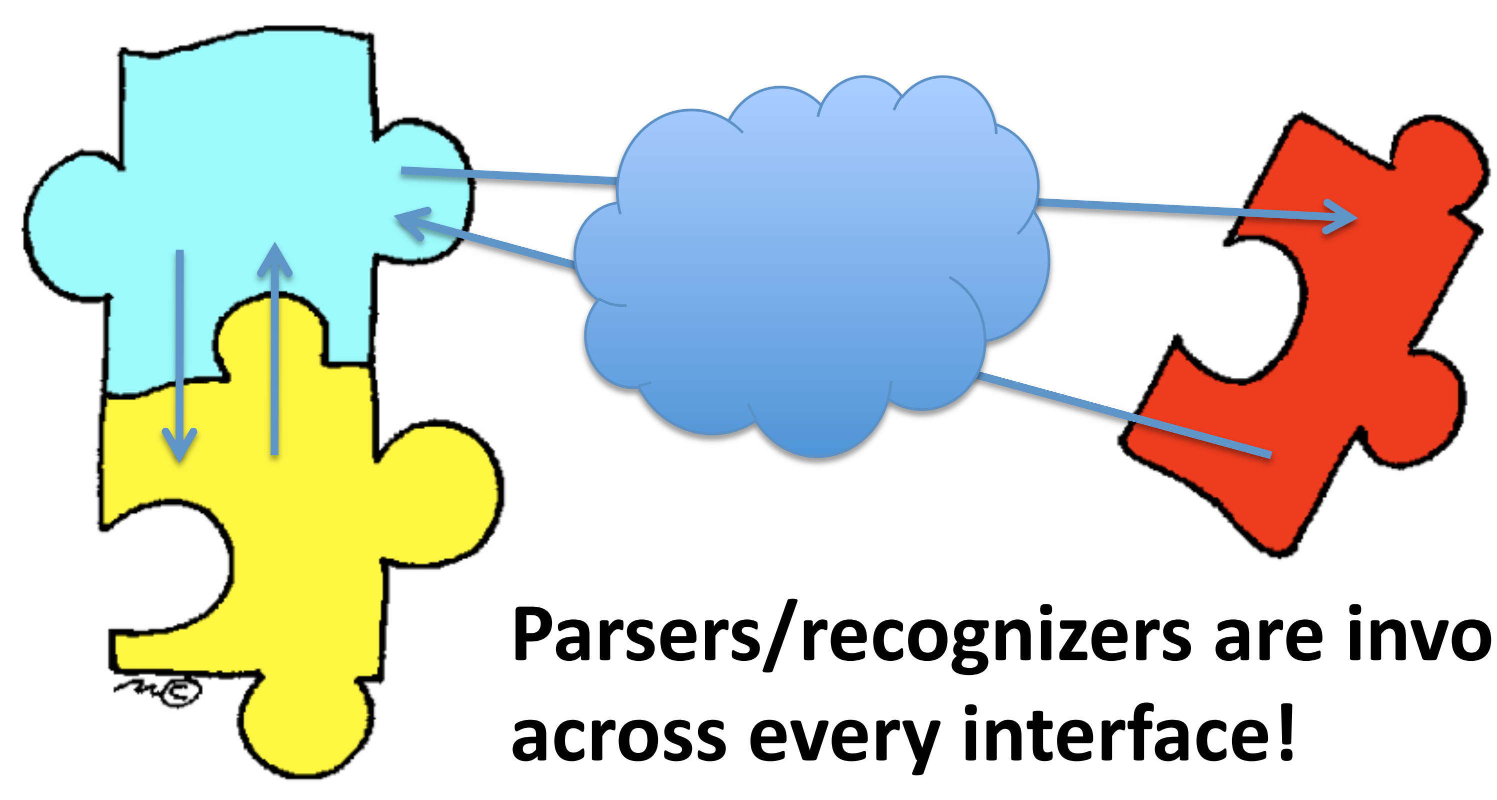
Computational equivalence between components:

"Are you seeing what I'm seeing?"



Insecurity: miscommunication

- Today's systems are distributed/composed, with many talking components



Parsers/recognizers are involved across every interface!

Parser computational equivalence

- Parsers involved in a protocol/exchange must parse messages **exactly the same way**
  - For X.509 SSL certs between CA and browser, formally required
  - Between a NIDS and its protected target, effectively required

- Equivalence must be assured/**tested**
  - with automation tools, unit tests, integration tests

The X.509 Case Study

- X.509’s Common Names (CN) :
  an ambiguous language, ad-hoc parsers =>
  - Certificate Signing Request (CSR) parsed differently by the signing CA and certificate consumer (e.g., browser) =>
  - Browser believes the CA signed this cert for google.com, ebay.com, paypal.com, …
- 20+ 0-day from Parser Differential Analysis
  - Sassaman, Patterson “Exploiting Forest with Trees”

Halting Problem, hello again

- Testing <u>computational equivalence</u> for two automata recognizing **regular** languages (regular expressions) and **deterministic pushdown** automata is **decidable**
  - Tools/software automation can help

- **But for non-deterministic pushdown automata or stronger it is UNDECIDABLE**
  - No amount of automated testing effort will give reasonable coverage

The curious case of the **IDS:** moving the Halting Problem around

- Trying to “fix” **Input Recognition** Halting Problem of a scattered and vaguely defined recognizer with another, “less vulnerable” component?
  - – But it can’t be fixed! So a “fix” **must** backfire.

- So you get the **Endpoint Computational Equivalence** Halting Problem between the IDS’ stack and the target’s input protocol handling!

"Insertion, Deletion, Evasion" & other horsemen of the Digital Apocalypse

- Ptacek & Newsham, 1998
- Vern Paxson, 1999--…

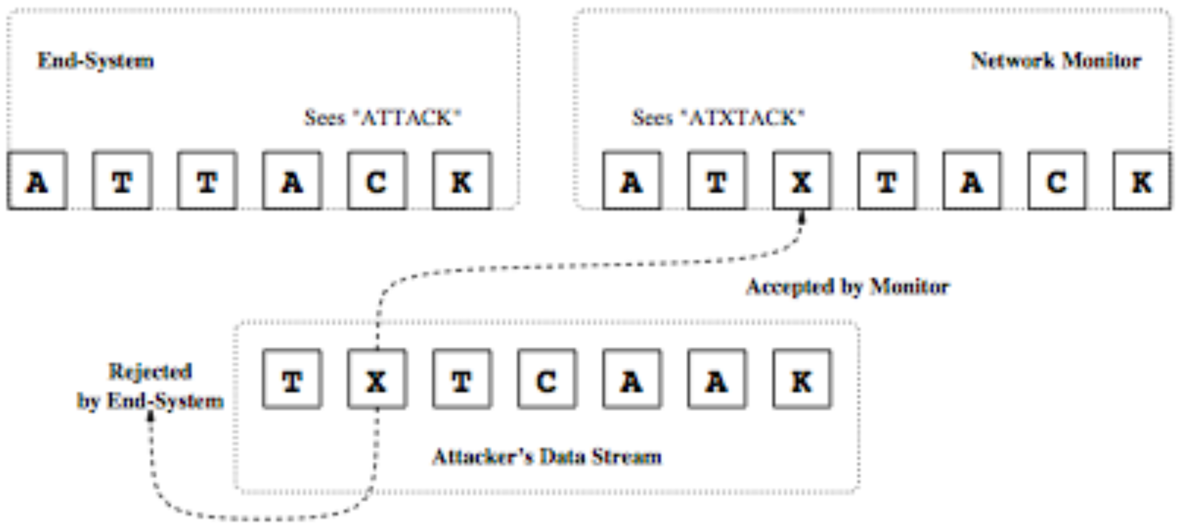


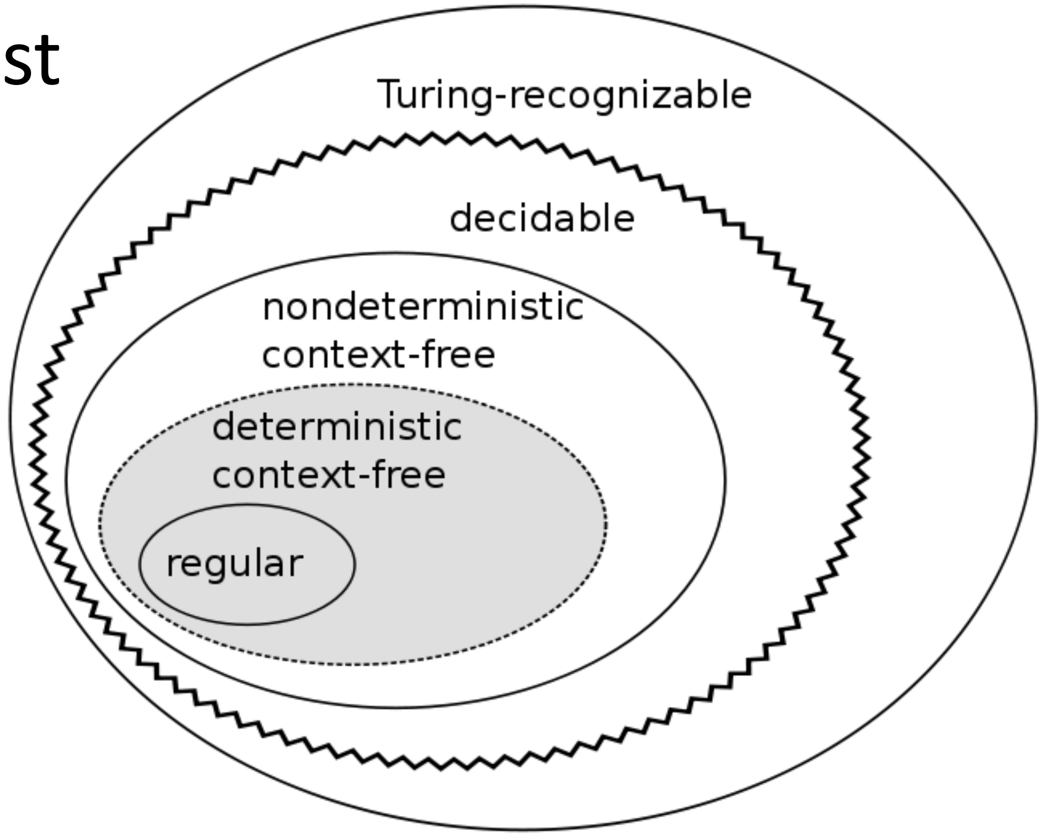
Figure 4: Insertion of the letter 'X'

“Conservation of (bad) computational power”

- Computational power once created cannot be destroyed
- “Dark energy” of scattered parsers will resurface
- You have not fixed HP due to input language complexity, you just converted it into another HP

Stay away from the Halting Problem

- Choose the input language **as simple as possible**, preferably **regular** or at most **deterministic context-free**



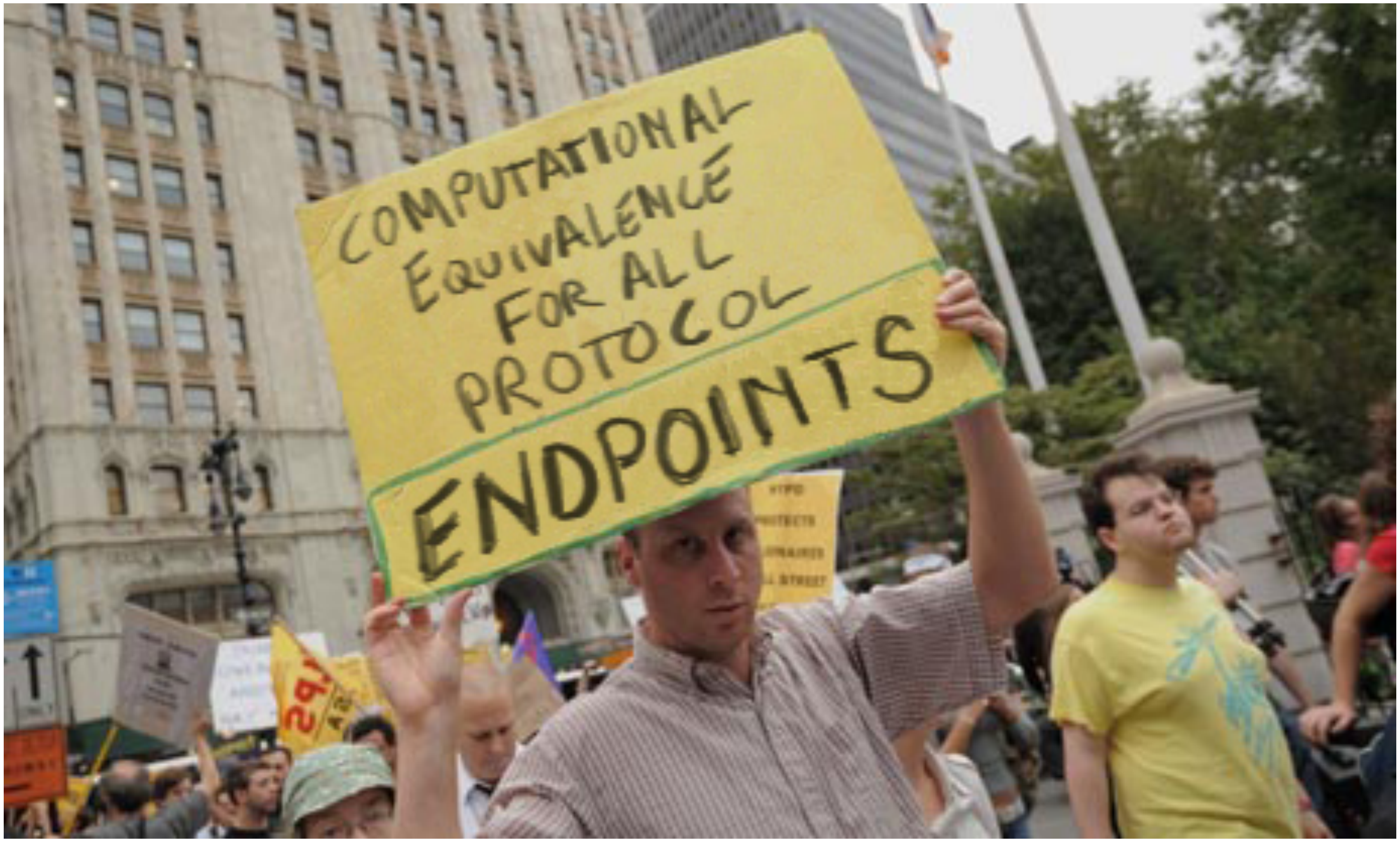
Time to re-evaluate Postel's Law?

"Be conservative in what you send;
be liberal in what you accept."

-- it made the Internets happen and work

- Sassaman & Patterson, PhNeutral Berlin March 2010
- Dan Geer, "Vulnerable Compliable" ;login: December 2010

Occupy the IETF!



Take-away?

- Good protocol designers don’t allow their protocols to grow up to be Turing-complete
- Ambiguity is Insecurity!
- If your application relies on a Turing-complete protocol, it will take infinite time to secure it
- Rethink Postel’s Law

Do not mistake complexity for functionality

- Saving money on future upgrades thanks to Turing-complete "extensibility"?

  See that you are not going to lose more on security/mediation/controls, eaten up by the Turing Beast.

- "This system is very extendable/updatable because it embeds macros/scripting/ programming language in data" -- run like hell

Money talks

Language-theoretic approach helps to

1. save mis-investment of money and effort,
2. expose vendors that claim security based on solving perpetual motion,
3. pick the right components and protocols to have manageable security,
4. avoid system aggregation/integration nightmare scenarios.

Occupy Input Handling!

- “Stop Weird Machines”
- “No More Turing---complete Input Languages!”
- “Reduce Computing Power Greed!”
- “No processing before recognition!”
- “Full recognition before processing!”
- “Comp equiv for all protocol endpoints!”
- “Context-free or Regular!”

Thank you!

http://langsec.org