The Science of Insecurity

Len Sassaman, K.U. Leuven
Meredith L. Patterson, Red Lambda
Sergey Bratus, Dartmouth College

TROOPERS12
Make the world a safer place.

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

This talk in 1 minute

- Huge share of insecurity comes from **protocol** and **message format** designs that, to get processed **securely**, require solving provably UNSOLVABLE problems.
- Designers/implementors set themselves up to strive against a **law of nature**, and so keep increasing attack surface.
- It's <u>not</u> hard to stop doing this: **think (simple) language theory when handling inputs.**

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 …

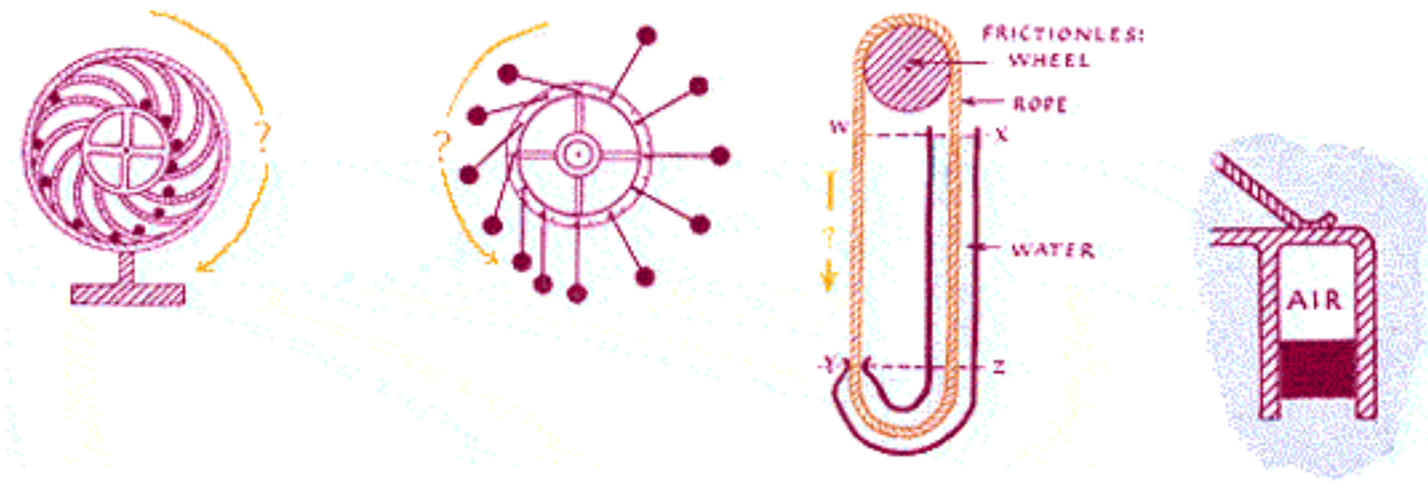
- Yet software still sucks!
- And hardware – we don’t even know how much it sucks (no tools to poke its attack surface -- yet)

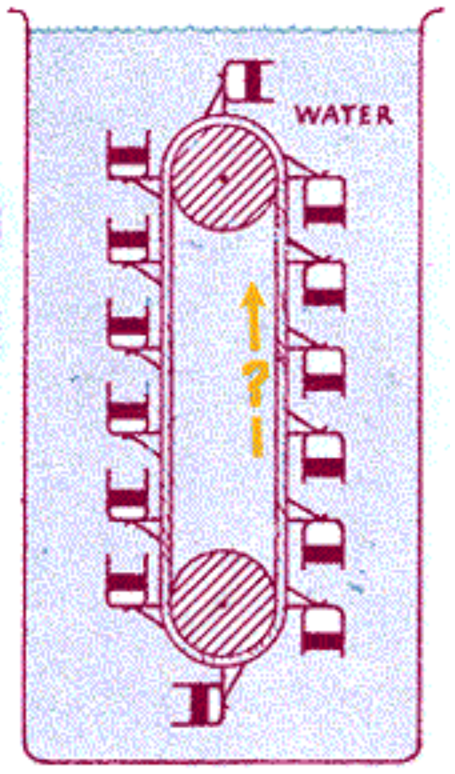
The Internet is here: ubiquitous pwnage

CO$ FORCES ITS FEMALE MEMBERS TO GET ABORTIONS
WHY ARE THEY DEAD?
LONGCAT IS LOOOOOOOOOOOOOOOOOOOONG!
THE CHURCH OF SCIENTOLOGY KILLS
KNOWLEDGE IS
SCIENTOLOGY MAKES ME A SAD PANDA
HONK
OH FUCK

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”

What **is** INsecurity?

- Holes for sneaking in executable code?
  – Nah, not since ROP (hackers 2000, academia 2007)
- Memory corruption?
- In-band signaling?
- Exposing unnecessary privilege?
- All of the above?

Wikipedia on Causes of Vulnerabilities

Vulnerability (computing)

- Complexity
- Familiarity
- Connectivity
- Password management flaws
- Fundamental OS flaws
- Internet Website Browsing
- Software bugs
- Unchecked user input
- Not learning from past mistakes

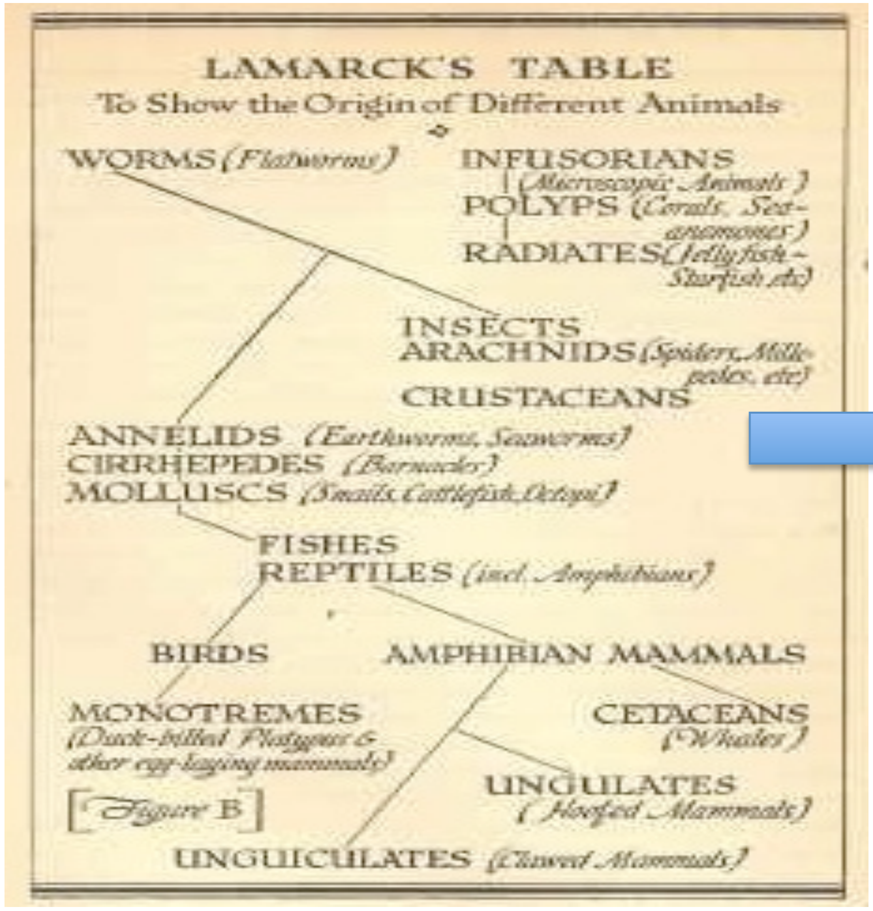
Vulnerability classifications?

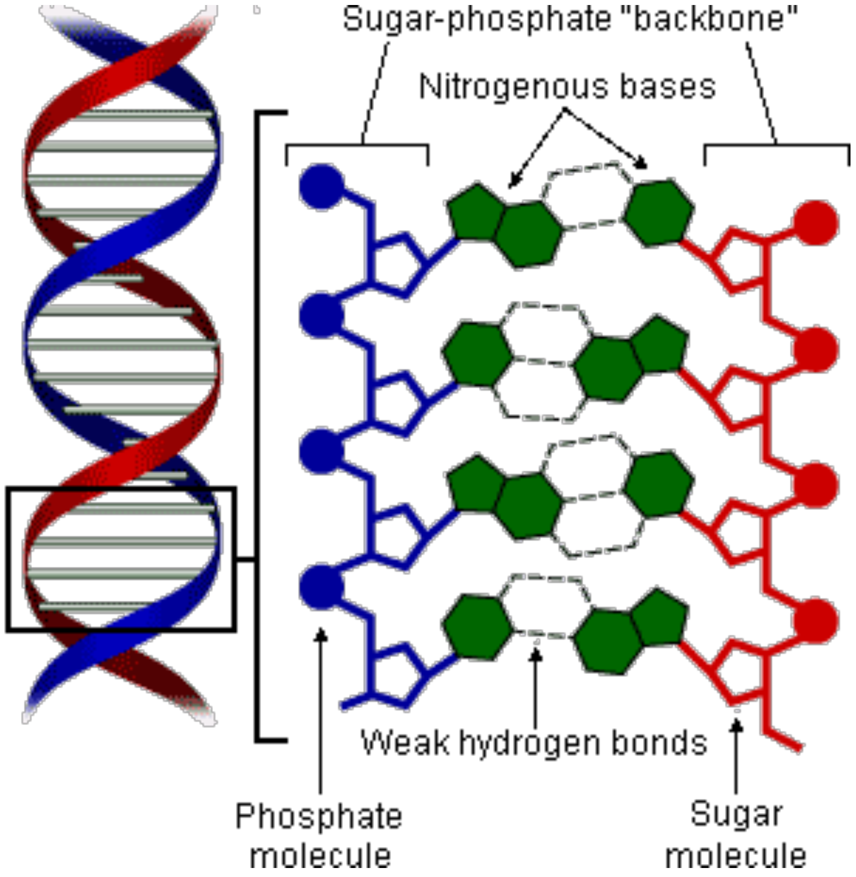
[In] a certain Chinese encyclopaedia … the animals are divided into:
(a) belonging to the emperor, (b) embalmed,
(c) tame, (d) suckling pigs,
(e) sirens, (f) fabulous, (g) stray dogs,
(h) included in the present classification,
(i) frenzied, (j) innumerable,
(k) drawn with a very fine camelhair brush,
(l) others, (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”:
Computational cladistics of vulnerability


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)
[Figure B]
UNGUICULATES (Clawed Mammals)



Insecurity is about **computation**

- *Trustworthiness* of a computing system is **what the system can and cannot compute**
  - Can the system **decide** if an input is invalid/unexpected/malicious & ***safely reject*** it?
  - Can it be trusted to **never** do X, Y, Z?

- Exploitation is **unexpected computation** caused reliably or probabilistically by some (crafted) **inputs**
  - *See **langsec.org/** for our exploits history sketch*

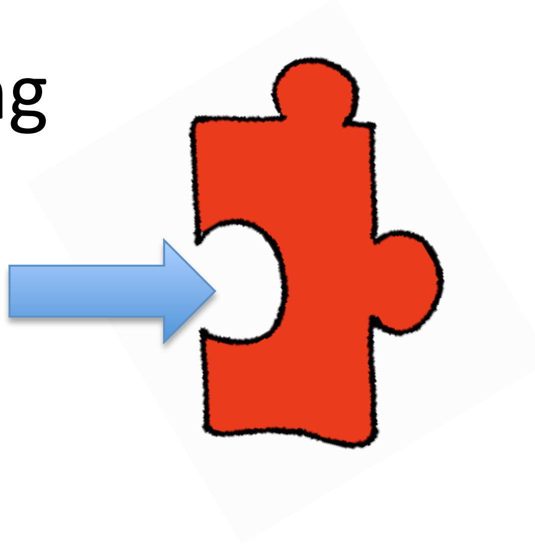
“Is this input good?“/ “Can this input hurt?”

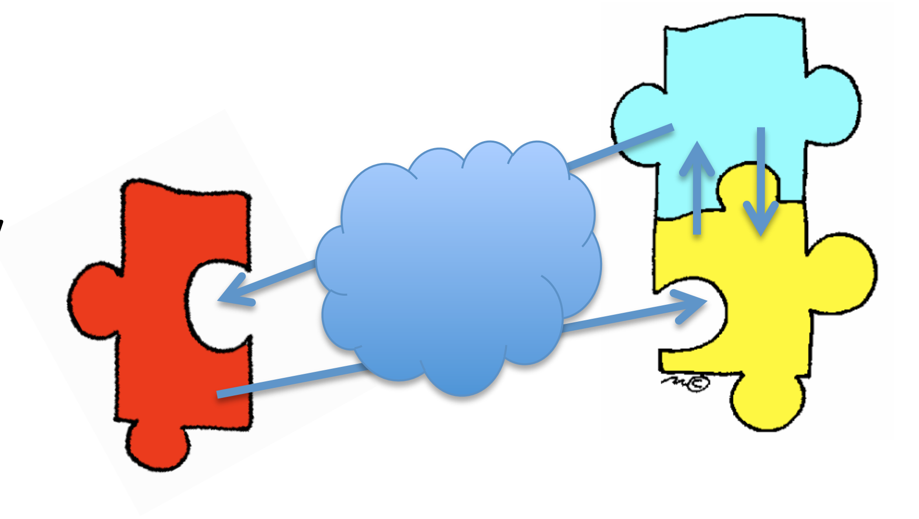
- Computation has some **unsolvable** (**undecidable**) problems – about **recognition of inputs**!

- **Undecidable problem:** an algorithm that would solve it in general **cannot exist**

Basic requirements in a **composed** world

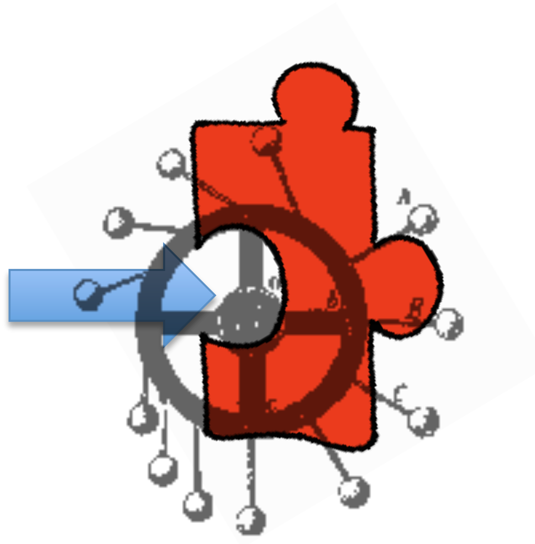
- One component/program accepting inputs
  - Must **accept** or **reject** messages safely
- Components communicating (distributed system, layered architectures)
  - Messages must be interpreted **identically** by endpoints



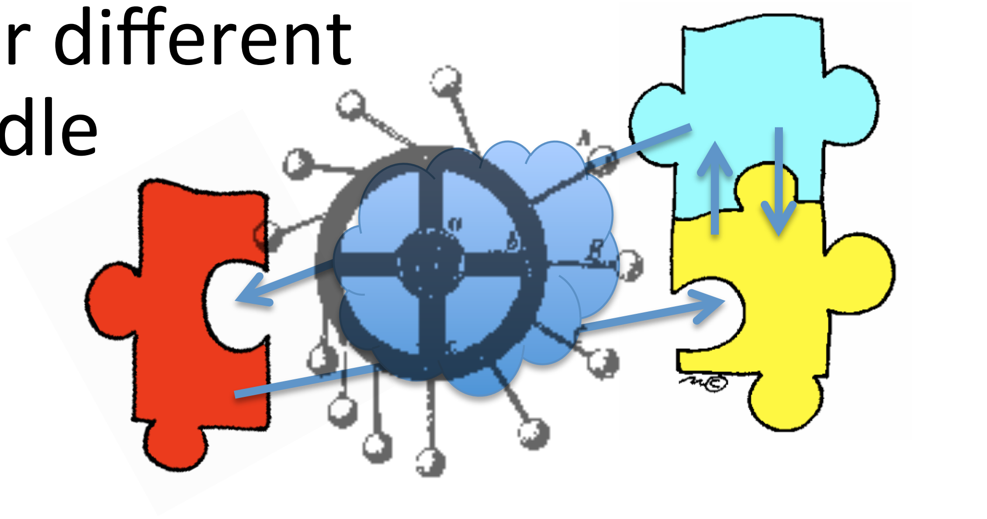


Undecidable Problems Attack!

- Some message/file formats are so complex that telling “good”/valid inputs from “bad”/invalid ones is **undecidable**



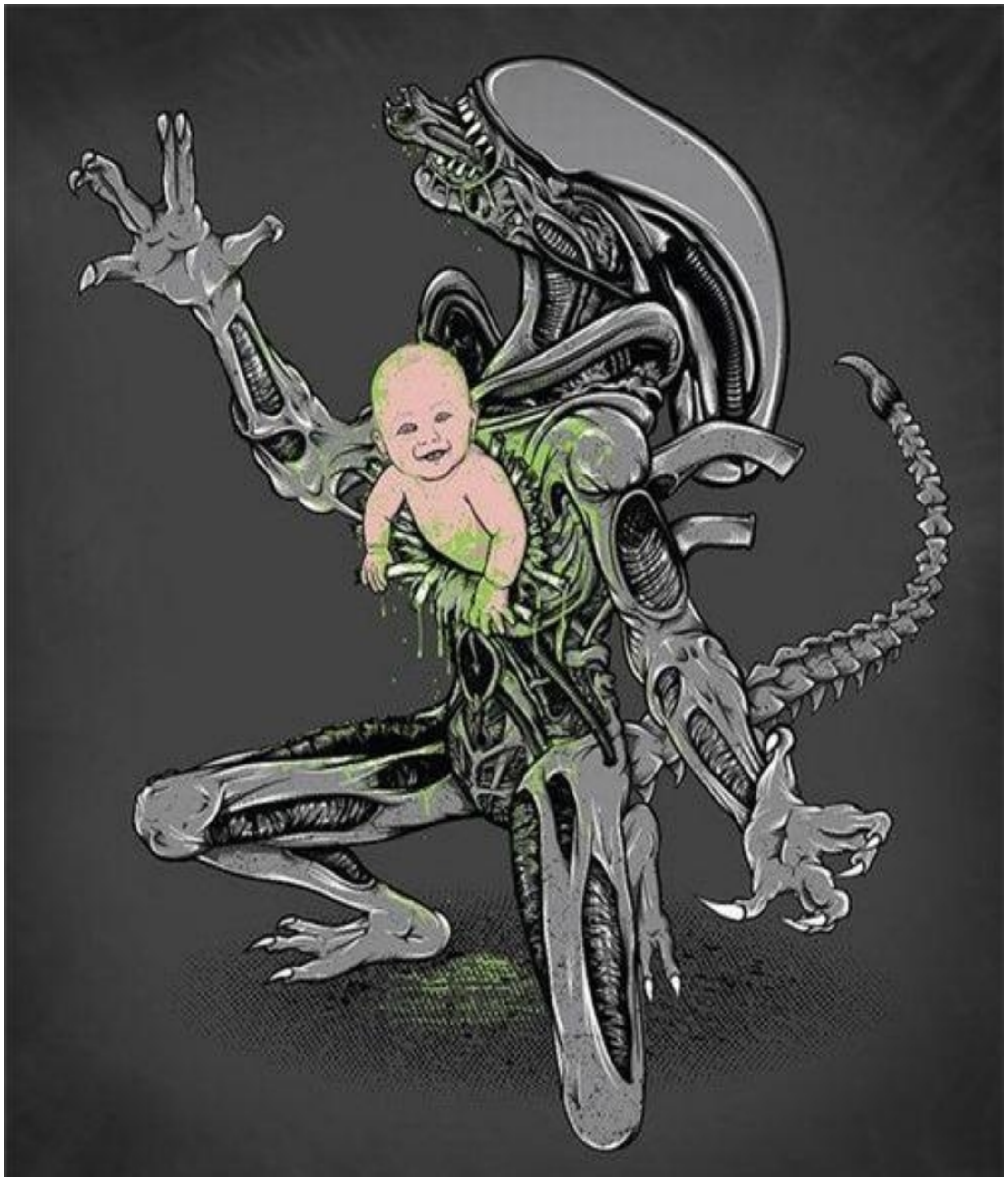
- Some protocols are so complex that checking whether different implementations handle them equivalently is **undecidable**



What happens when input recognition fails?

- What internal code gets is **not** what it expects
- **Primitives** are exposed
  - Memory corruption, implicit data flow
  - Unexpected control flow, … <you know it>
- A “**weird machine**” is born
  - A **more powerful**, programmable **execution environment** than intended or expected

“A weird machine is born”



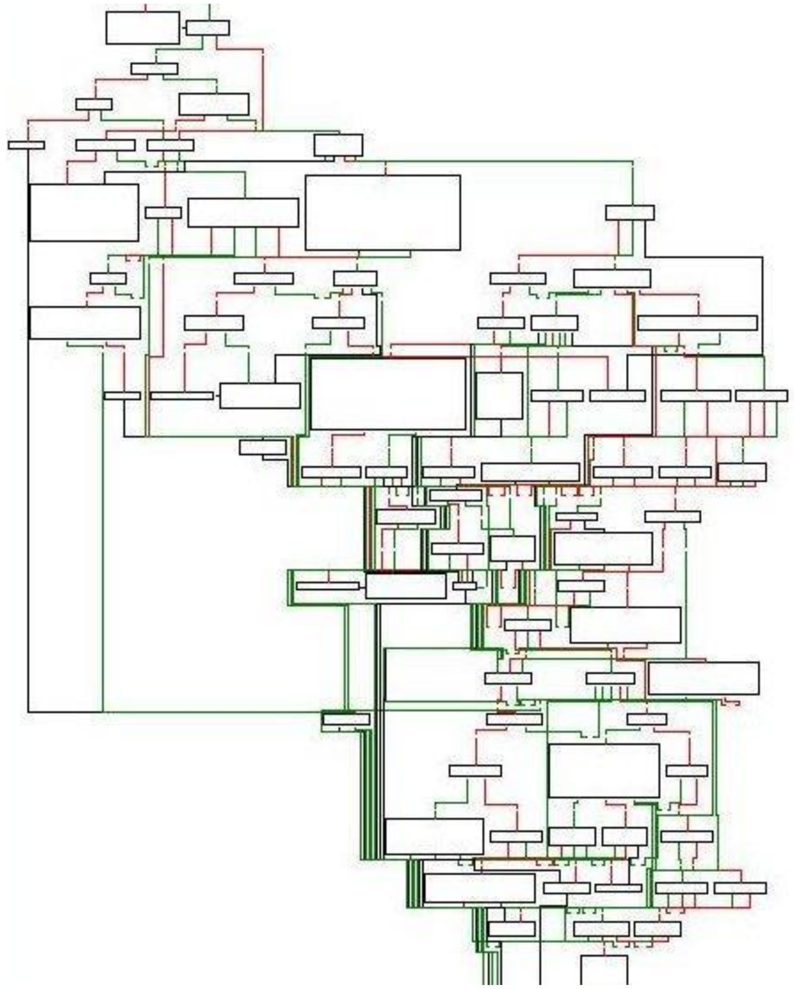
“Exploitation is setting up, instantiating, and programming a weird machine” – *Halvar Flake, Infiltrate 2011*

- A part of the target is overwhelmed by crafted input and enters an **unexpected** but **manipulable** state
- **Exploit** is a program for WM, written in crafted input
- **Inputs** drive the unexpected computation that runs on WM

“The Hidden/Scattered Recognizer” a.k.a. “Shotgun Parser”

- Checks for input validity are scattered throughout the program, mixed with processing logic

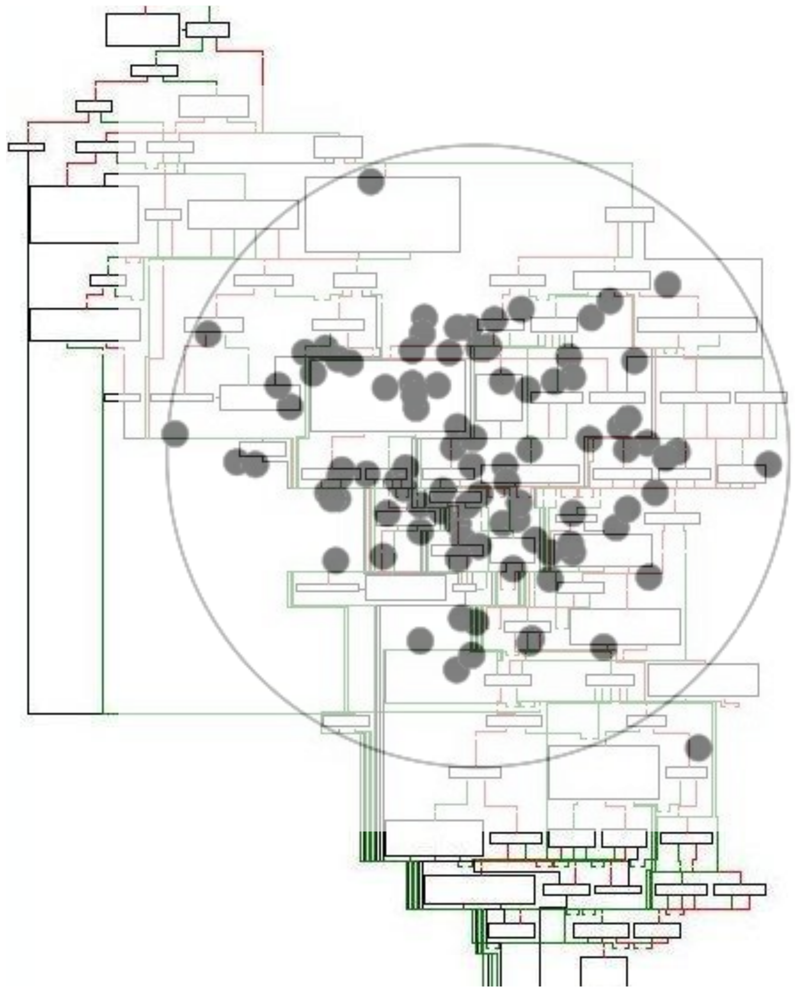
- Ubiquitous, deadliest programming/”design” pattern



“The Hidden/Scattered Recognizer” a.k.a. “Shotgun Parser”

- Checks for input validity are scattered throughout the program, mixed with processing logic

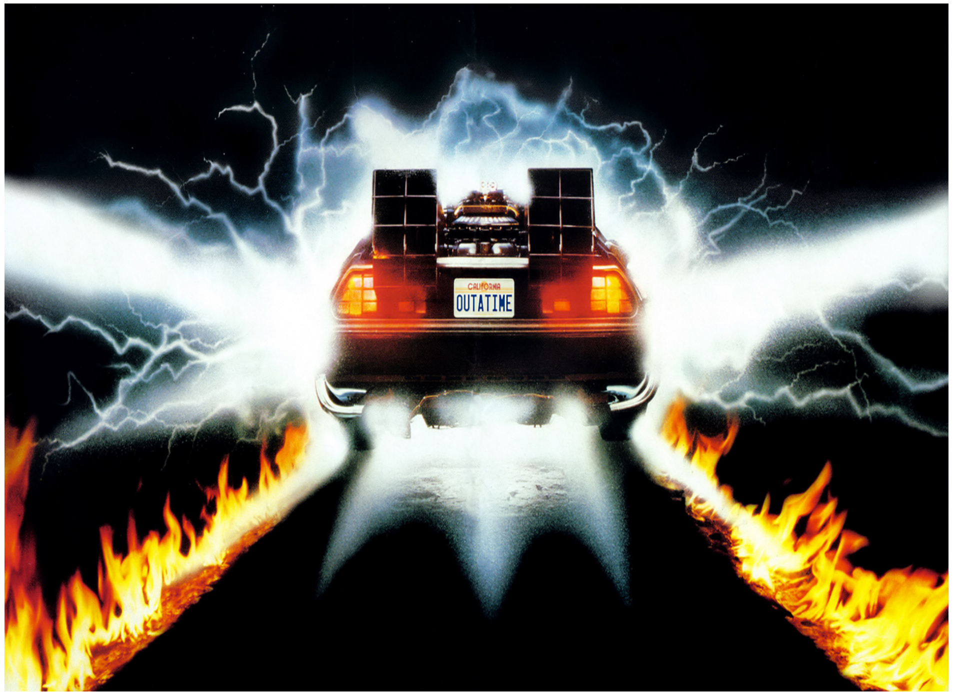
- Ubiquitous, deadliest programming/”design” pattern



Insecurity is Unexpected Computation

- Academics study **models of computation**
- Hackers study actual **computational limits** of **real systems**, expose hidden computation architectures hidden in targets
- “It’s not a **bug**, it’s a **cog** of a Weird Machine!”

Back to the Turing Future to slay the Turing Beast!



Back to **Church** and **Turing** to understand exploitation

The history of the Uncomputable

- Leibniz: “*Can a machine determine truth values of mathematical statements*”? [17th century]

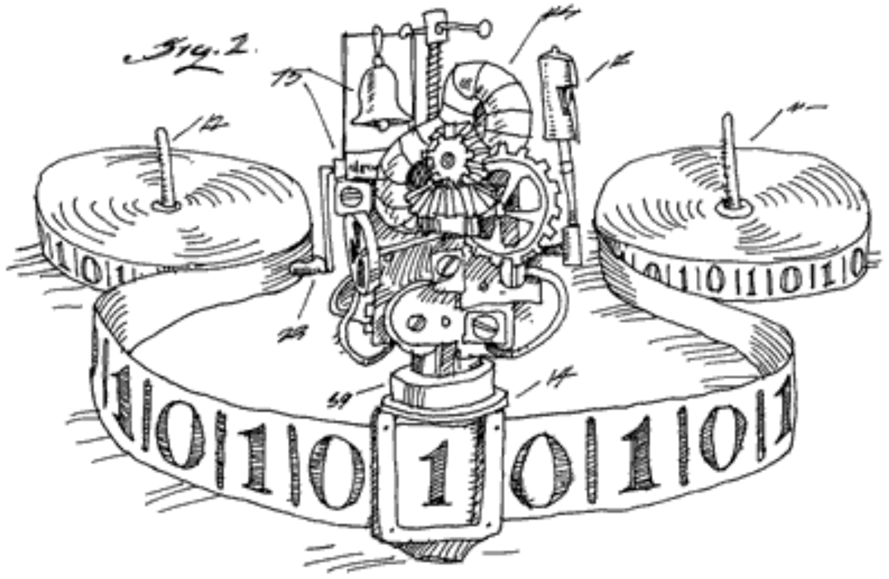
- Hilbert’s Entscheidungsproblem, [1928]
  - “*Is an arbitrary logical statement valid or invalid*”?

- Church [1936], Turing [1937]: ***Negative!***
  - Based on work by Kleene, Goedel [1930s]

Turing machines and undecidability

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

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

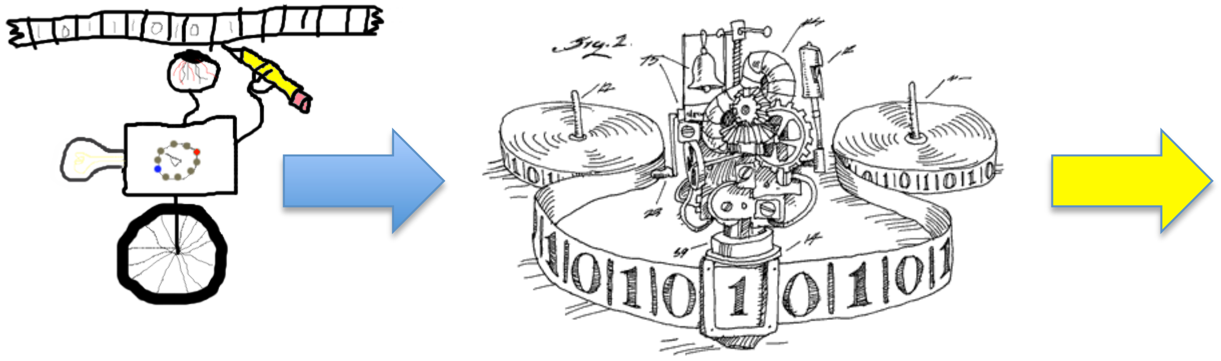


Undecidable problems: No TM can solve them.

"The Halting Problem is Undecidable"

Cornerstone: the Halting Problem

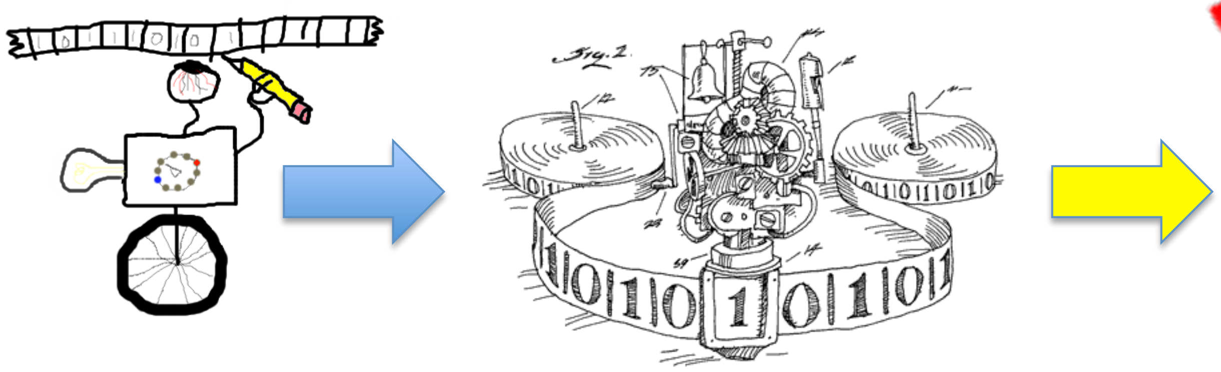
- “I can build a TM that takes another TM as input and decides if it will ever terminate”



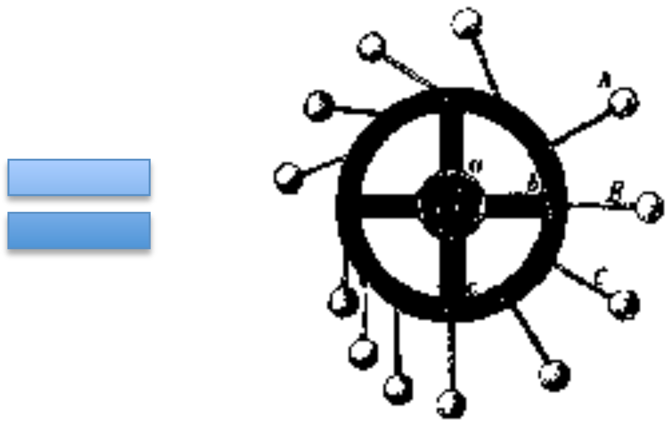
{True, False}

Cornerstone: the Halting Problem

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



{True, False}

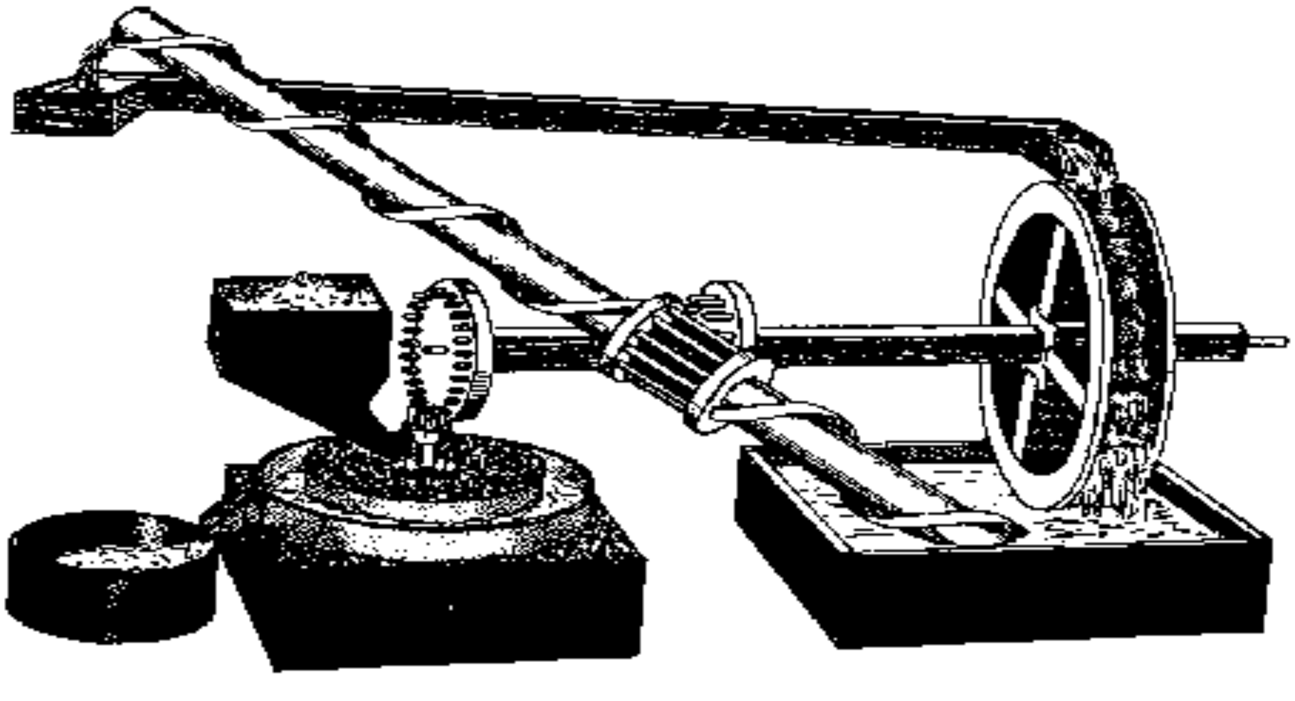


Some designs force programmers of input recognizer to “solve” the UNDECIDABLE Halting Problem

- **Halting Problem** w.r.t. **inputs** and **communications** (protocols, formats):
- Bad news: no amount of **testing** or “**fixing**” will help
- Good news: they can be avoided

There is no “80/20” engineering solution for the Halting Problem

- Same as for Perpetual Motion
- If someone is selling it, run away

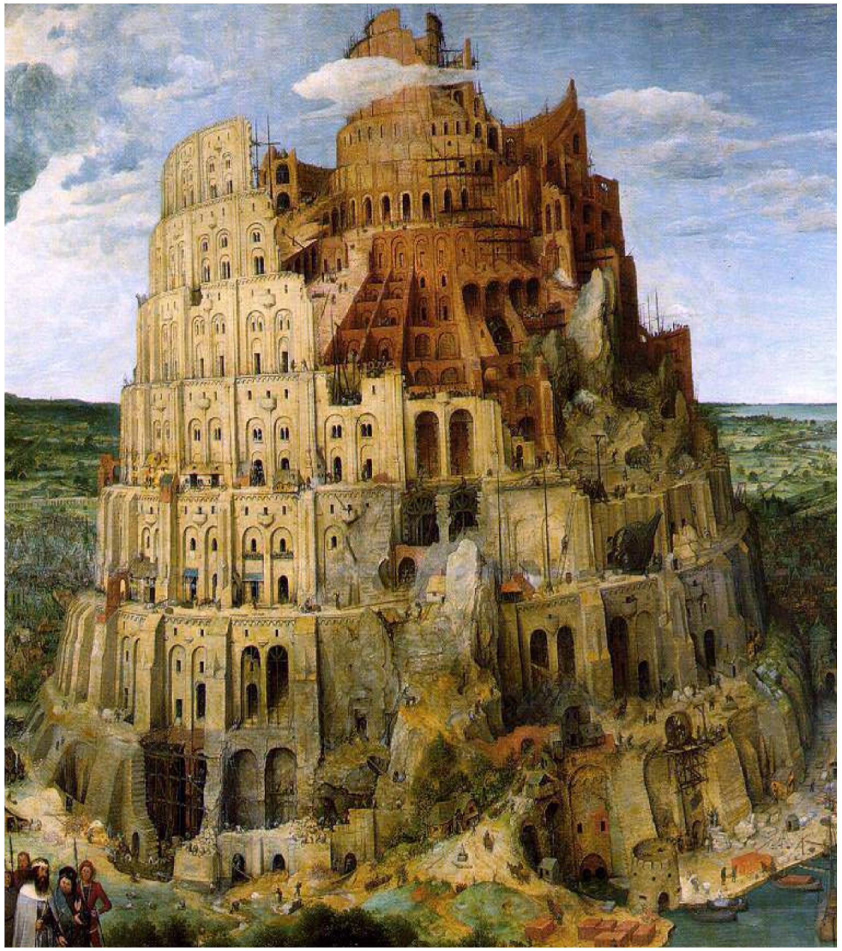


All Bytes Are Created Equal

- But we want them to have different meanings
- So we **wrap** them in other bytes
- We also **nest** the wrappings
- Trouble starts when unwrapping is hard
  - Need to know too many bytes to interpret a byte
  - 1858-2324-5331-2811-5903
    4840-3907-7331-7064-0747

Languages vs Computation

- **Inputs are a language**
  - as in "formal language"
- Some languages are **much harder** to recognize than others
- For some, recognition is **undecidable**

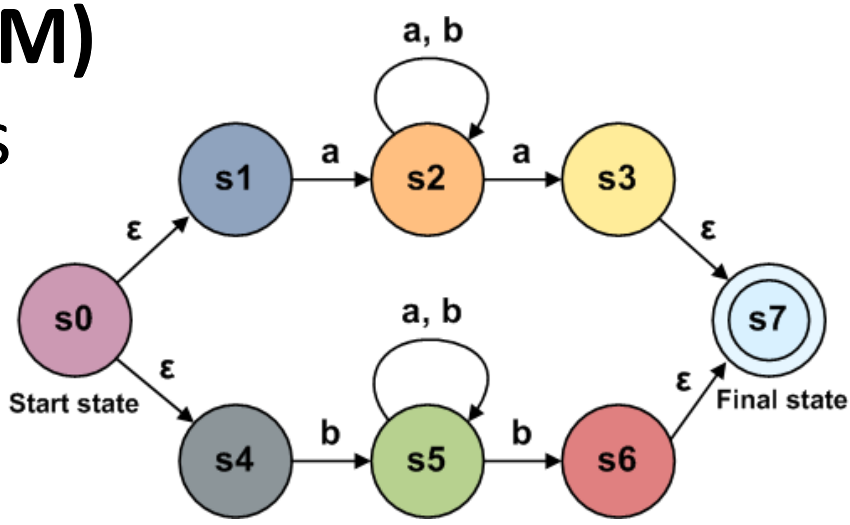


Languages are everywhere!

- Network stacks: valid **packets** make a language
  - Stack is a recognizer at every layer/protocol
- Servers: valid **requests** make a language
  - e.g. SQL injection is a recognizer failure
- Memory managers: **heaps** make a language
  - Heap metadata exploits abuse its context-sensitivity
- Function call flow: valid **stacks** make a language
  - Context-sensitivity again, which bytes are data, which are metadata?

Regular Languages

- **Finite state machines (FSM)**
  Simple nesting, delimiters



Ex.: a[ab]+a | b[ab]+b

Note: Matching **recursively nested structures** with Regexps will **fail**

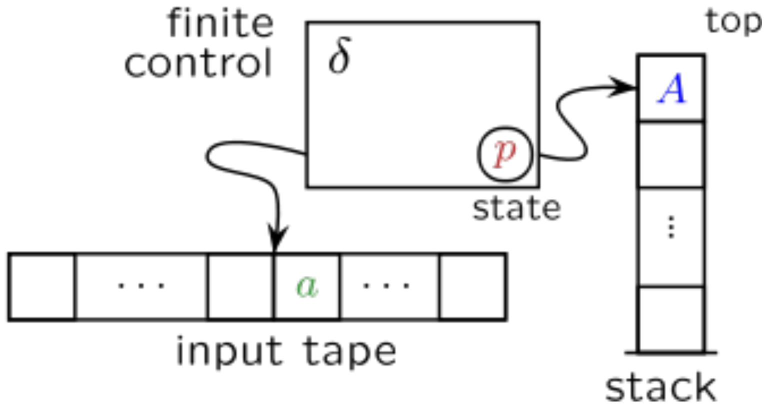
- (((([(({((…)})])))) , XML, HTML, anything with unlimited escaping levels, …

Context-free Languages

- Matching recursively nested structures:

  pushdown automata

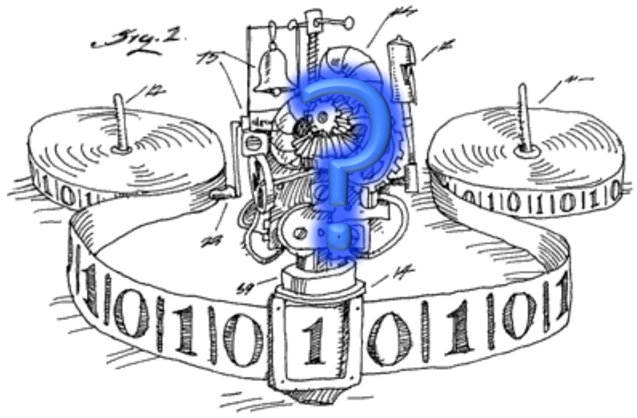
  (FSM + stack)



Ex.: Arbitrary depth of balanced parentheses
  ((([({(…)})]))), S-expressions, …

Context-sensitive Languages

- Require a linear-bounded automaton (Turing machine with finite tape, still decidable)



Ex.: some metadata is needed to interpret the rest of the data

Ex.: protocols with **length fields** are ***weakly context-sensitive*** *(decidable)*

Think of parsing an IP packet past a few corrupted bytes

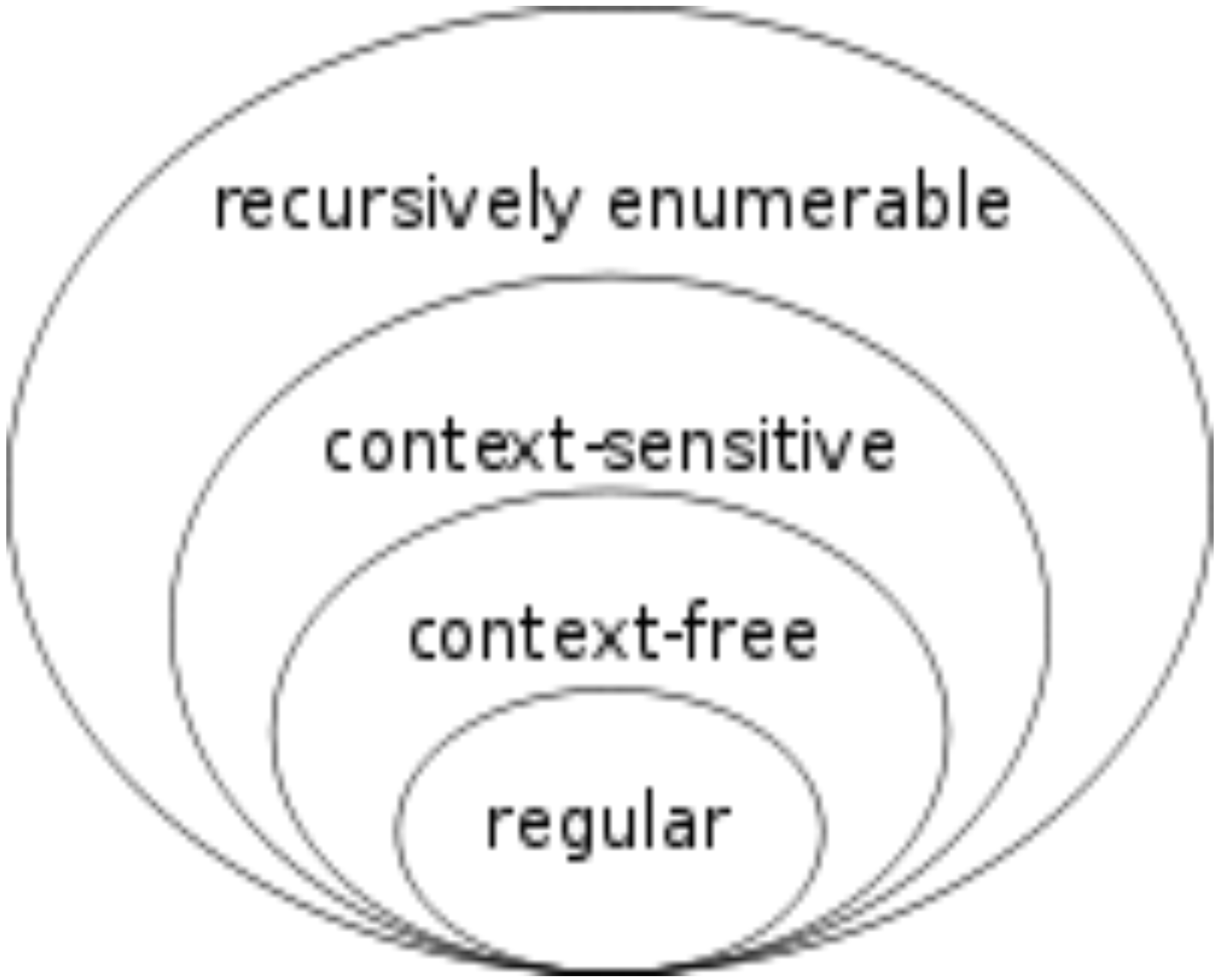
Turing-complete Languages

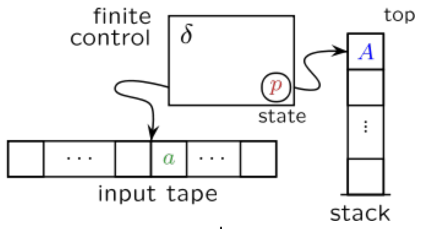
- Telling if input is a program that produces a given result: **UNDECIDABLE**

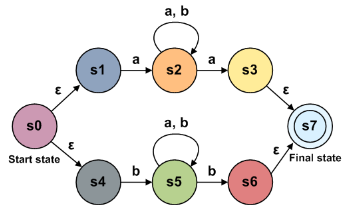
  (a.k.a. Rice’s Theorem)

Ex.: telling if any given code or message with macros/scripts is ‘*good*’ or ‘*malicious*’ without running it

The language hierarchy







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**
- If the **recognizer** does not match the **language**, it is broken
- If neither is well-defined or understood, the program is broken

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,** data flows, and **transitions** (hello “weird machine”!)
  - Weird machines run on borrowed state
  - (cf. Halvar’s Infiltrate 2011 talk)

- 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

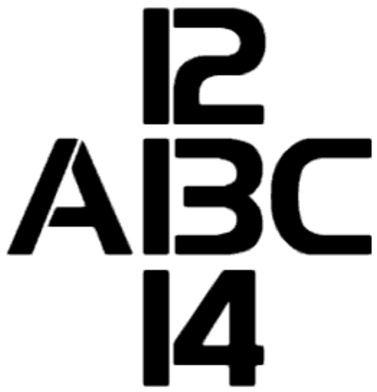
Occupy Message Formats!

Occupy Protocol Design!

II. Composition & communication

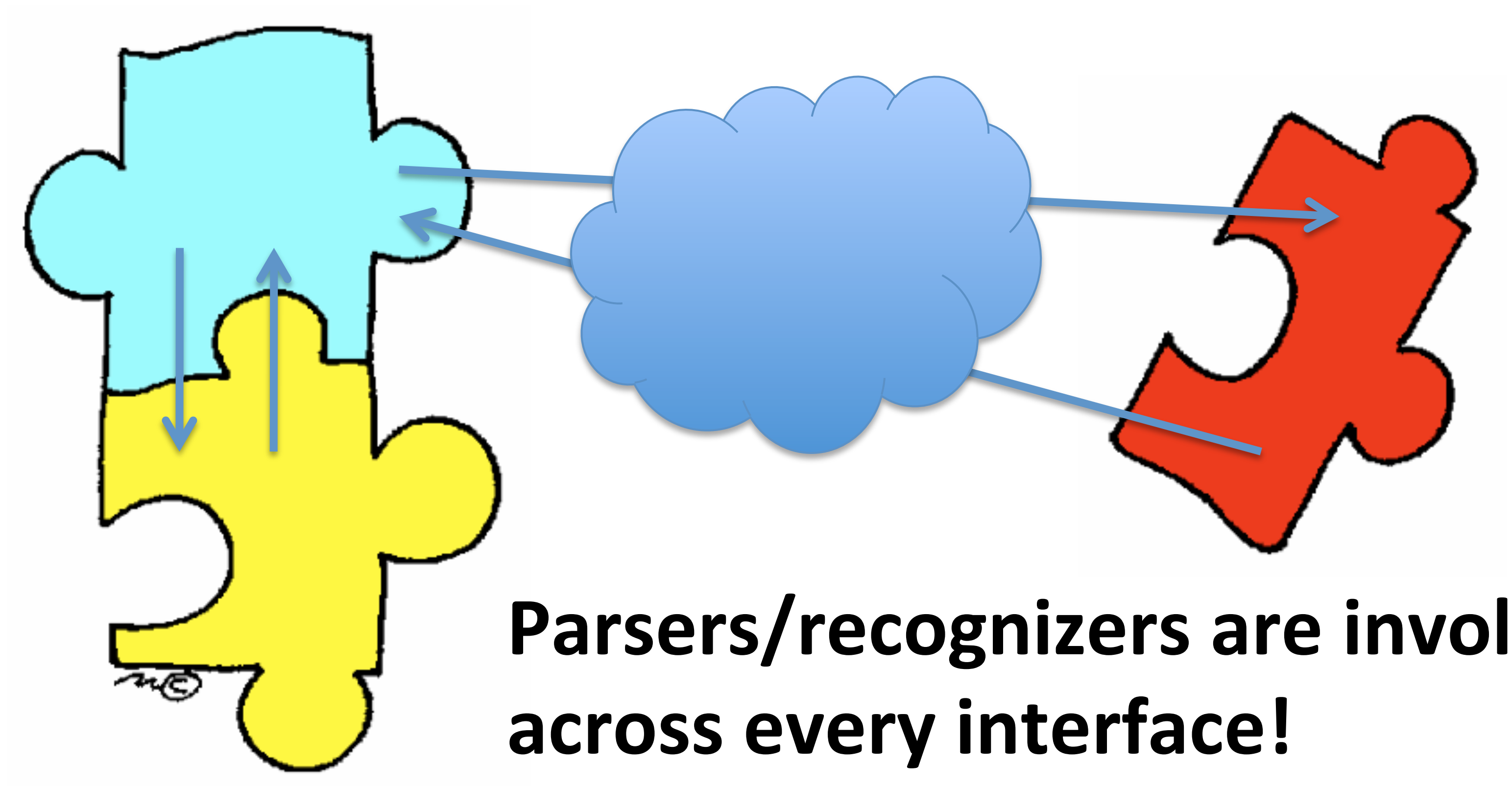
Computational equivalence between components:

“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 Parse Tree Differential Analysis
  - Sassaman, Patterson "Exploiting the Forest with Trees"
  - ASN.1 BER ambiguous, considered harmful

Hello halting, my old friend

- 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 IDS/IPS Apocalypse

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

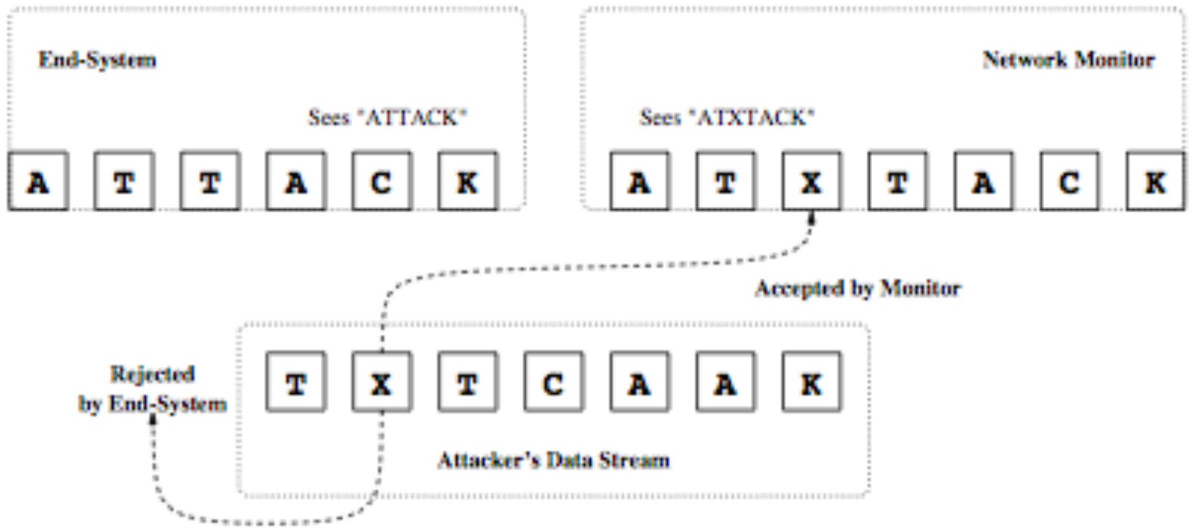


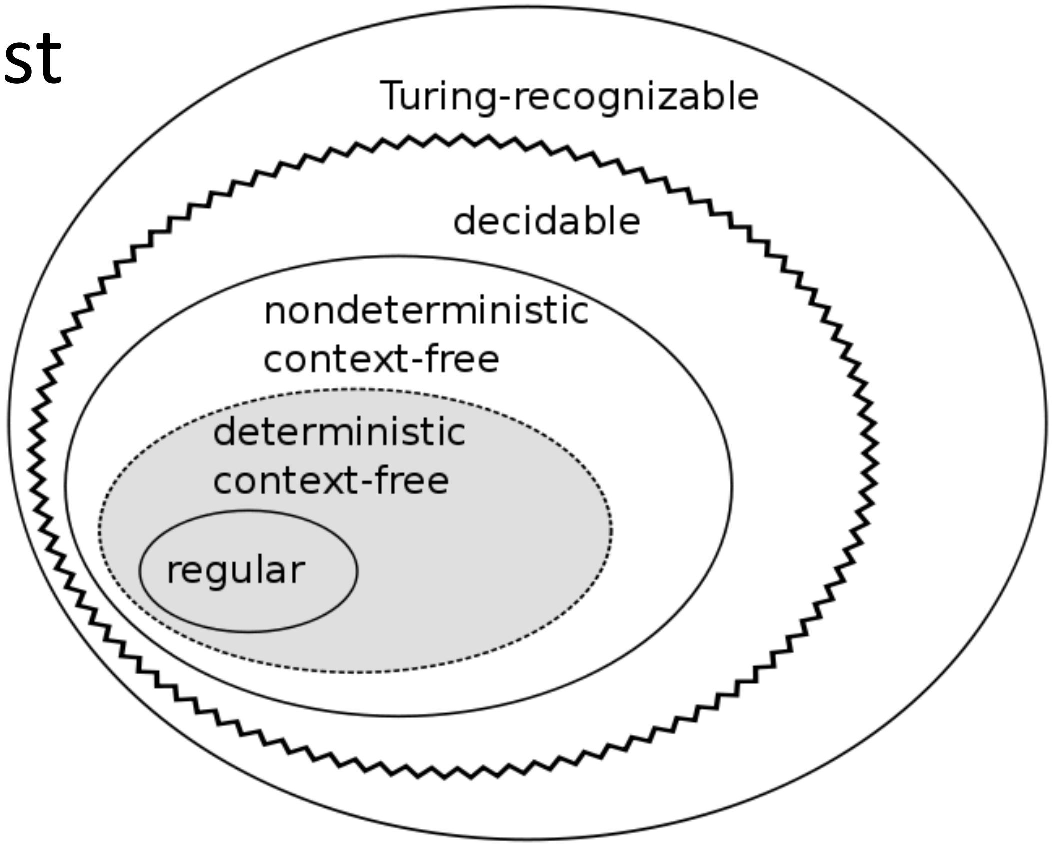
Figure 4: Insertion of the letter 'X'

“Conservation of (bad) computational power”

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

Stay away from the Halting Problem

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



Occupy the IETF!

Time to re-evaluate Postel's Principle?

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

-- it made the Internets happen and work

-- its misreadings made the Internets the way they are now

Postel's Principle needs a patch:

- Sassaman & Patterson, PhNeutral, March 2010
- Dan Geer, "Vulnerable Compliance" ;login: December 2010 (free online)

The Postel's Principle Patch

```
-  Be liberal about what you accept
+ Be definite about what you accept
+
+ Treat inputs as a language, accept it with a
+ matching computational automaton, generate its
+ recognizer from its grammar.
+
+ Treat input-handling computational power as
    privilege,
+ and reduce it whenever possible.
```

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

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!

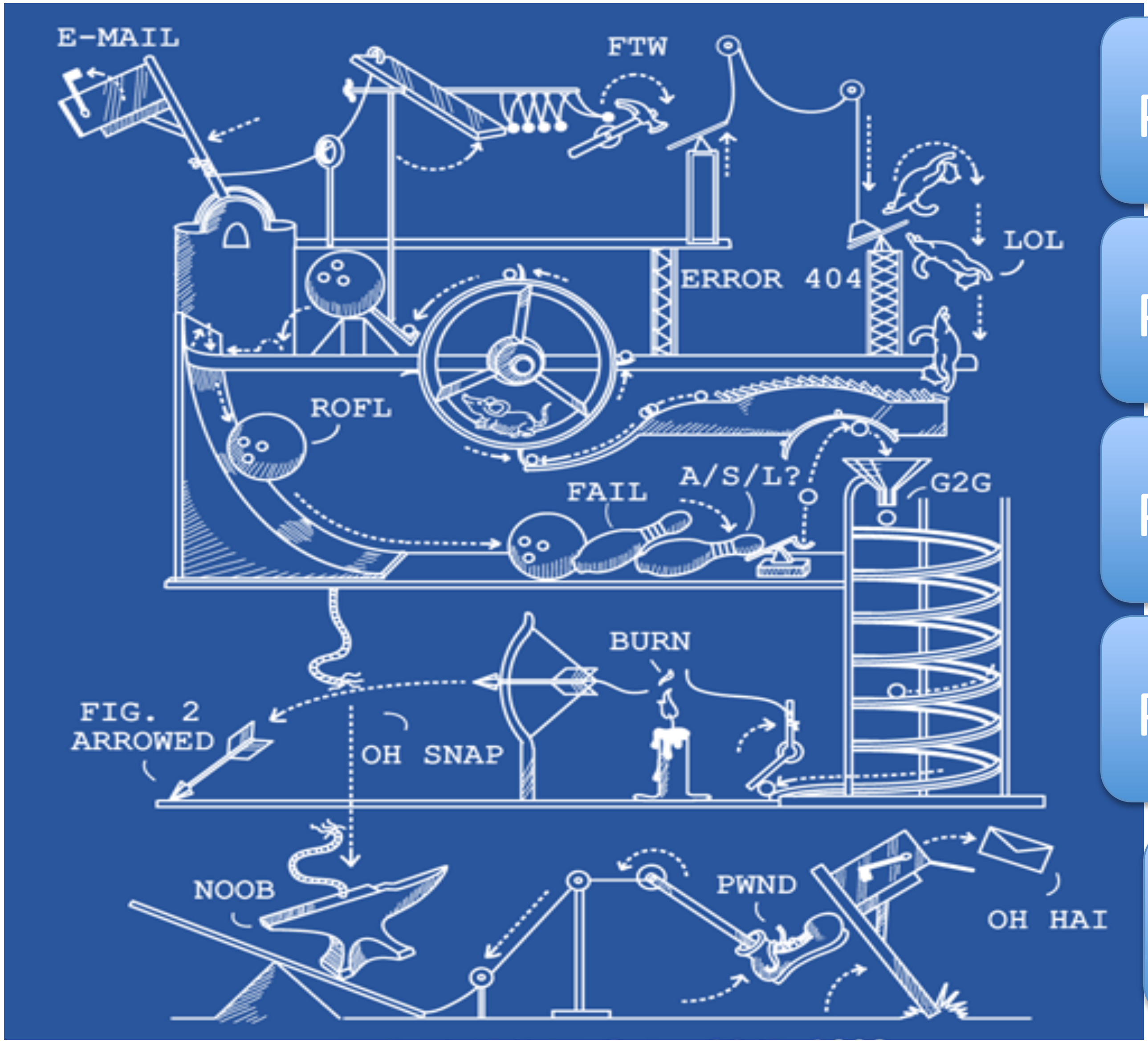
- “Stop Weird Machines”
- “No More Turing-complete Input Languages!”
- “Reduce Computing Power Greed!”
- “Ambiguity is Insecurity!”
- “Full recognition before processing!”
- “Computational equivalence for all protocol endpoints!”
- “Context-free or Regular!”

Thank you!

http://langsec.org

http://langsec.org/occupy/

"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