Return to the Hundred Acre Woods

what I've learnt in 3 years

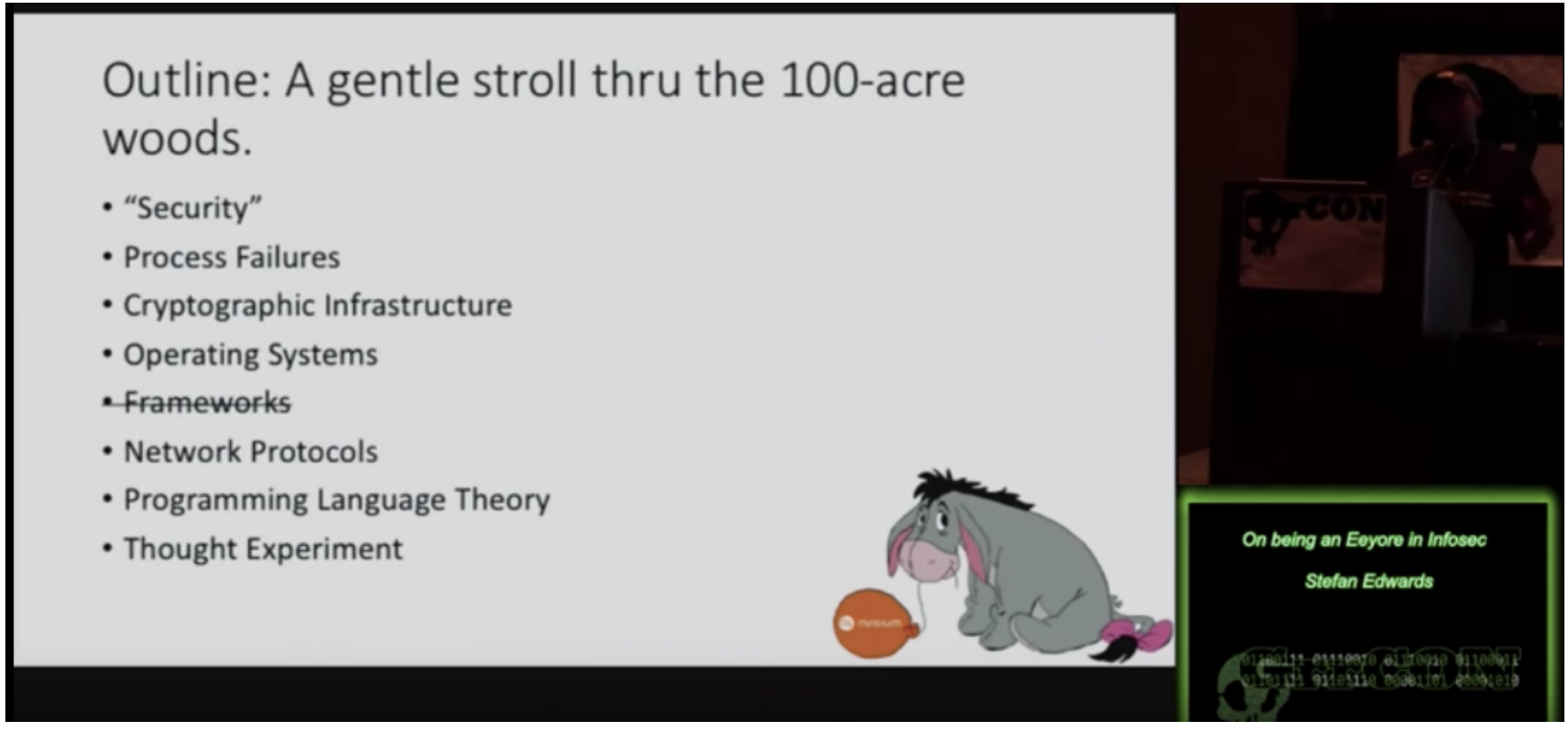
or

3eeyore5u

$ finger lojikil

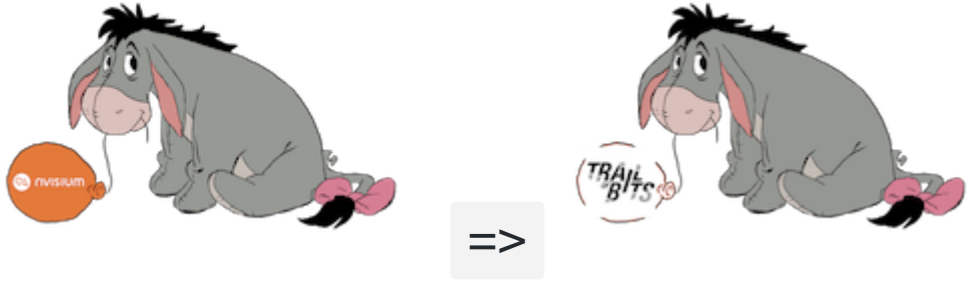
```
[lojikil.com]
Stefan Edwards (lojikil) is not presently logged in.

- Assurance Practice Lead, Trail of Bits
- Twitter/GitHub/Lobste.rs: lojikil
- Works in: Defense, FinTech, Blockchain, IoT, compilers,
vCISO services
- Previous: net, web, adversary sim, &c.
- Infosec philosopher, professional programming
language theorist, everyday agronomer, father.

WARNING: DEAF
WARNING: Noo Yawk
```

in 2016 I gave this talk



this talk

- cover what I've seen going from traditional infosec to high(er) assurance work
- talk about application of formal techniques & cryptography
- why none of this will save you, I'm still Eeyore



take aways

no golden roads to security

1. TINSTAAFL: formal tools & cryptography aren't panaceas
2. There is no golden road to ~~arithmetic~~ security
3. Formally proven code fails, fancy crypto breaks, OS setups can multiply problems

me: 2016 vs 2019

2016

- OSINT, threat modeling, blackbox, web, net, lots of gov, finance

2019

- those + formalisms, PLT, more exposure to fancy crypto

Me

2016

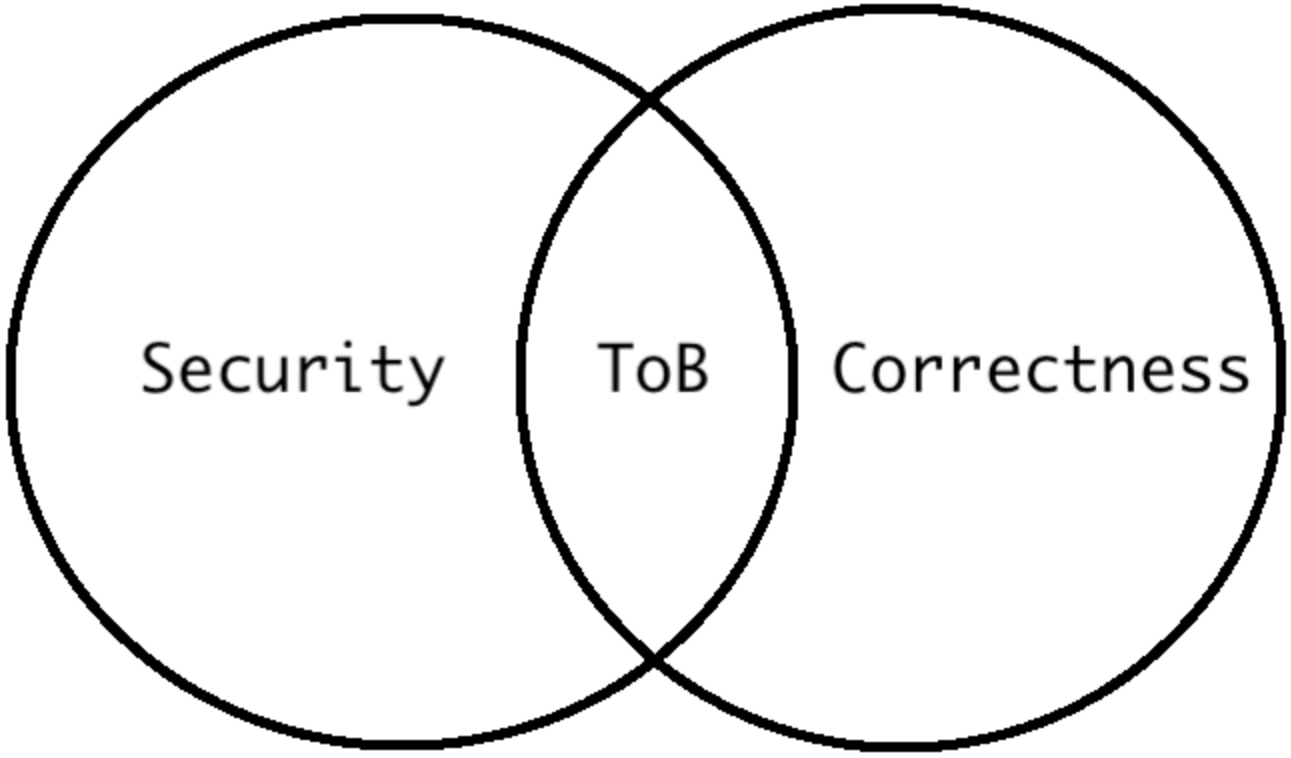
2019

when I said "security, I meant...

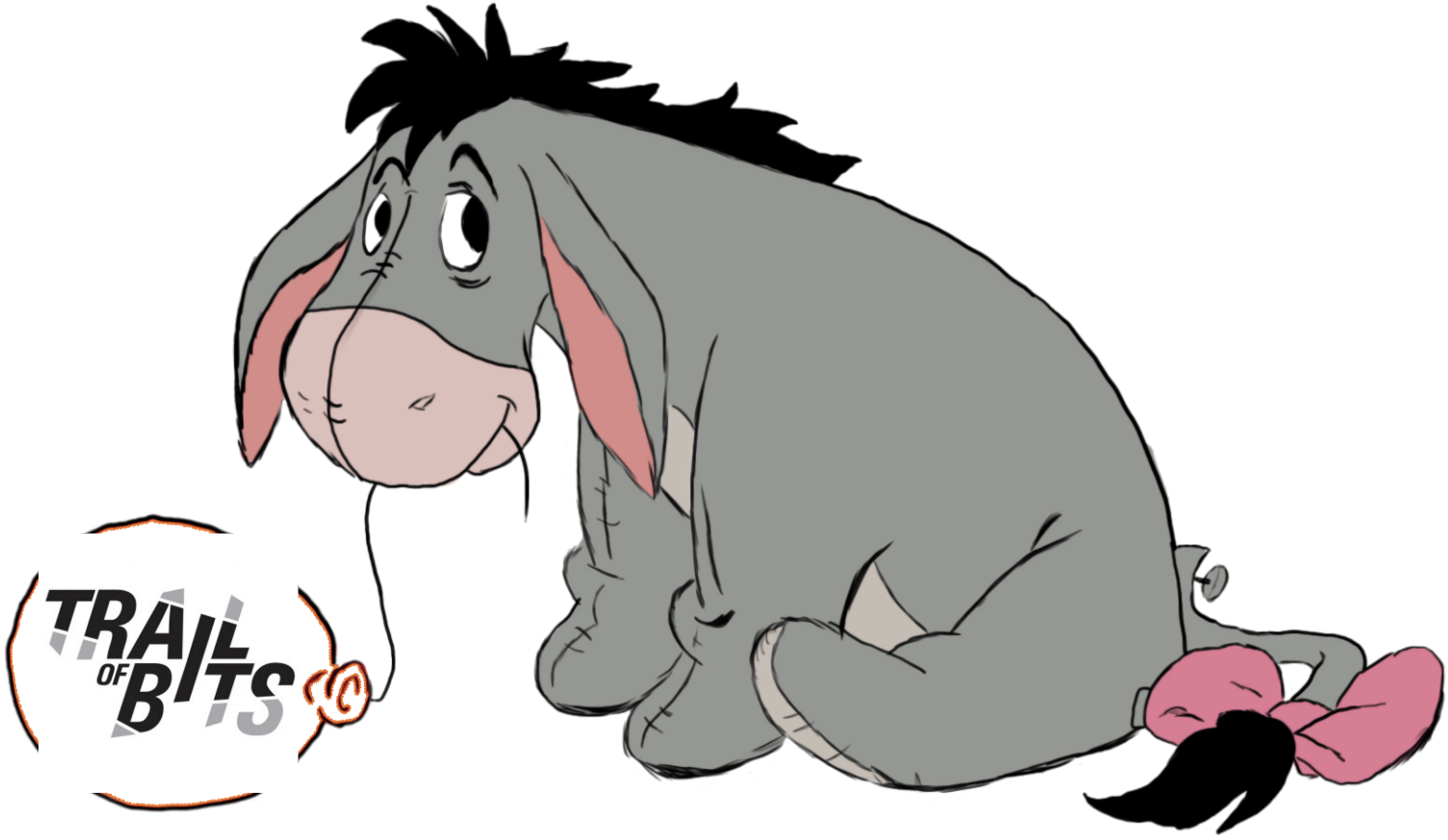
- design +
- denotation +
- formalized & verified model +
- centralized security controls +
- decentralized implementation ==
- "secure"

now when I say security...

- Not much has changed



let's update ourselves



formal tools won't save you

2016

- Java: Java Modeling Language (JML)
- C: Frama-C
- [fancy formal languages no one uses] : [fancy formal tools no one uses]
    - Eiffel, Sather, Lissac, any other DBC language there
    - TLA+, ACL2, &c were also alive
    - Ada has been around since forever

formal tools won't save you

- tools were verbose
- often unused, difficult to use, or unknown

a formal aside

- JML is > 20 years old (first specs & such from 1999)
- multiple compiler support it
- industry standard...
- "just" Hoare logic
- ... with large, longitudinal case studies...
- ... virtually unused

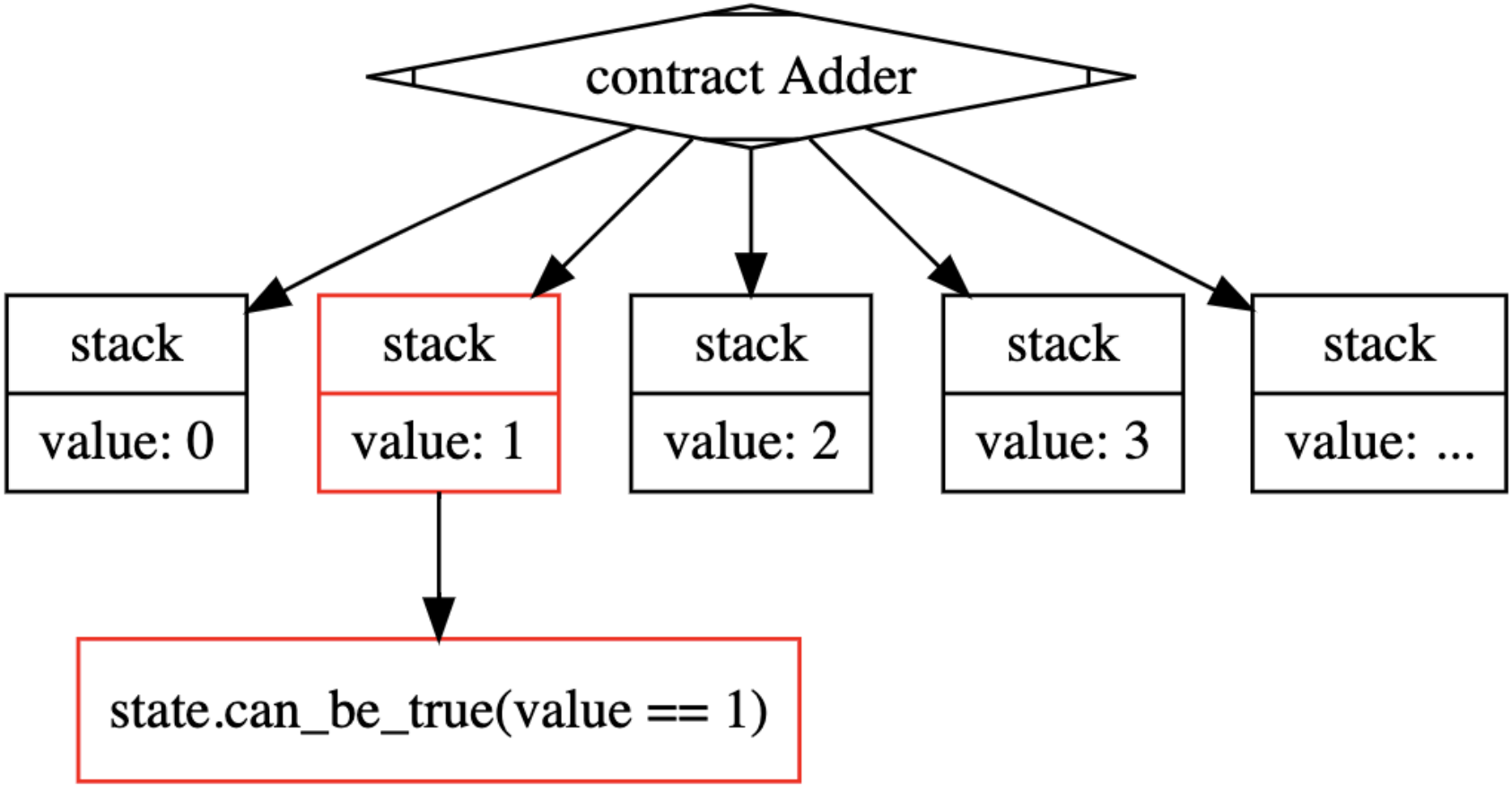
formal in 2019

- regularly interact with formal tooling
- since joining ToB, have written two, one for a client
- ease of use has gotten better
    - in Ethereum, there are *many* to chose from
- tend to lean towards ease of use vs specificity
    - lots of verifiers, less specifiers
    - more built in tests

formal in 2019

```
contract_src="""
contract Adder {
    function incremented(uint value) public
    returns (uint){
        if (value == 1)
            revert();
        return value + 1;
    }
}
"""
# ...
value = m.make_symbolic_value()

contract_account.incremented(value)
# ...
for state in m.ready_states:
    print("can value be 1? {}".format(
        state.can_be_true(value == 1)))
```

formal in 2019



formal in 2019

- pros:
  - much easier to start now
  - multiple implementations, geared towards devs, CI/CD
- cons:
  - the usual negatives
- as we'll see: formally proven/verified code *can* fail

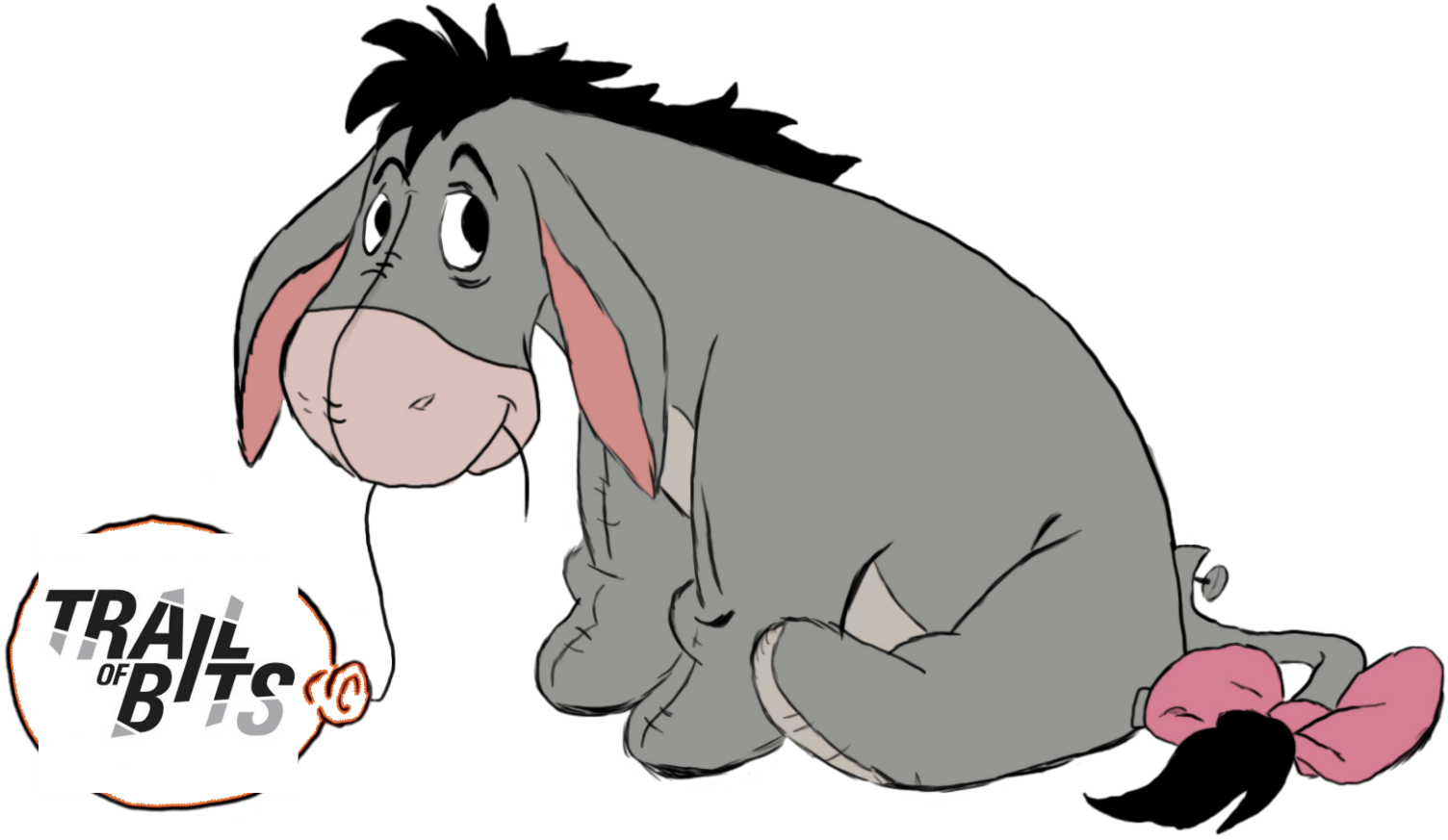
formally a failure

- only as good as your spec/verification
  - spec bugs
  - under spec
  - not spec'ing the right thing
- can't test everything

formally a failure

- tested code != prod code
  - Ariane 5 rocket
- things look positive
  - client proved with `uint`, then switched to `int`
- lots of blockchain examples

fuzzing won't save you



fuzzing contra symbex

from *http://deniable.org/reversing/symbolic-execution*

fuzzing in 2016

- SecLists
- Radamsa
- AFL
- QuickCheck (for Haskell), Hypothesis (Python)
  - available, but unused
- some exotic stuff like DART

https://patricegodefroid.github.io/public_psfiles/talk-pldi2005.pdf

fuzzing in 2019

- those, plus
- easily accessible grammar fuzzers
  - Mozilla Dharma, various others
- grey-box fuzzers
  - ECLIPSER
- other combinations of symbex + fuzzing
  - concretize via fuzzing
  - negate paths ala SAGE
- **property-based testing** is much more common

a fuzzy aside

- i still fine a ton of vulns with radamsa
- `radamsa -v -o $SOMEPATH` is so stupidly effective

fuzzing in 2019

- property-based testing is p popular now
- allows developers to write simple properties that must hold
  - `userIds <= 10000`
  - `fromAddress != toAddress`
- simple to write, often within the same language
- testing tool:
  i. generates random calls to API w/ random data
  ii. shrinks random call graph to minimum
  iii. all to find combinations that violate properties

fuzzing in 2019

```
contract TEST is NewCoin {
  uint private initSupply;
  address private alice = ...;
  address private bob = ...;
  address private eve = ...;

  constructor() public {
    balances[alice] = 10000;
    balances[bob] = 10000;
    balances[eve] = 10000;
    initSupply = totalSupply_;
  }
  // the actual good stuff:
  function echidna_test() public returns (bool) {
    totalSupply_ = balances[alice] + balances[bob] +
    balances[eve];
    return (initSupply == totalSupply_);
  }
}
```

fuzzing in 2019

- pros:
  - can find many edge cases
  - tools have become lightweight & easier to control
- cons
  - unlikely to randomly generate complex formats
  - anything > pure random takes time
- (more) ideal, combine constraint gen with fuzzing

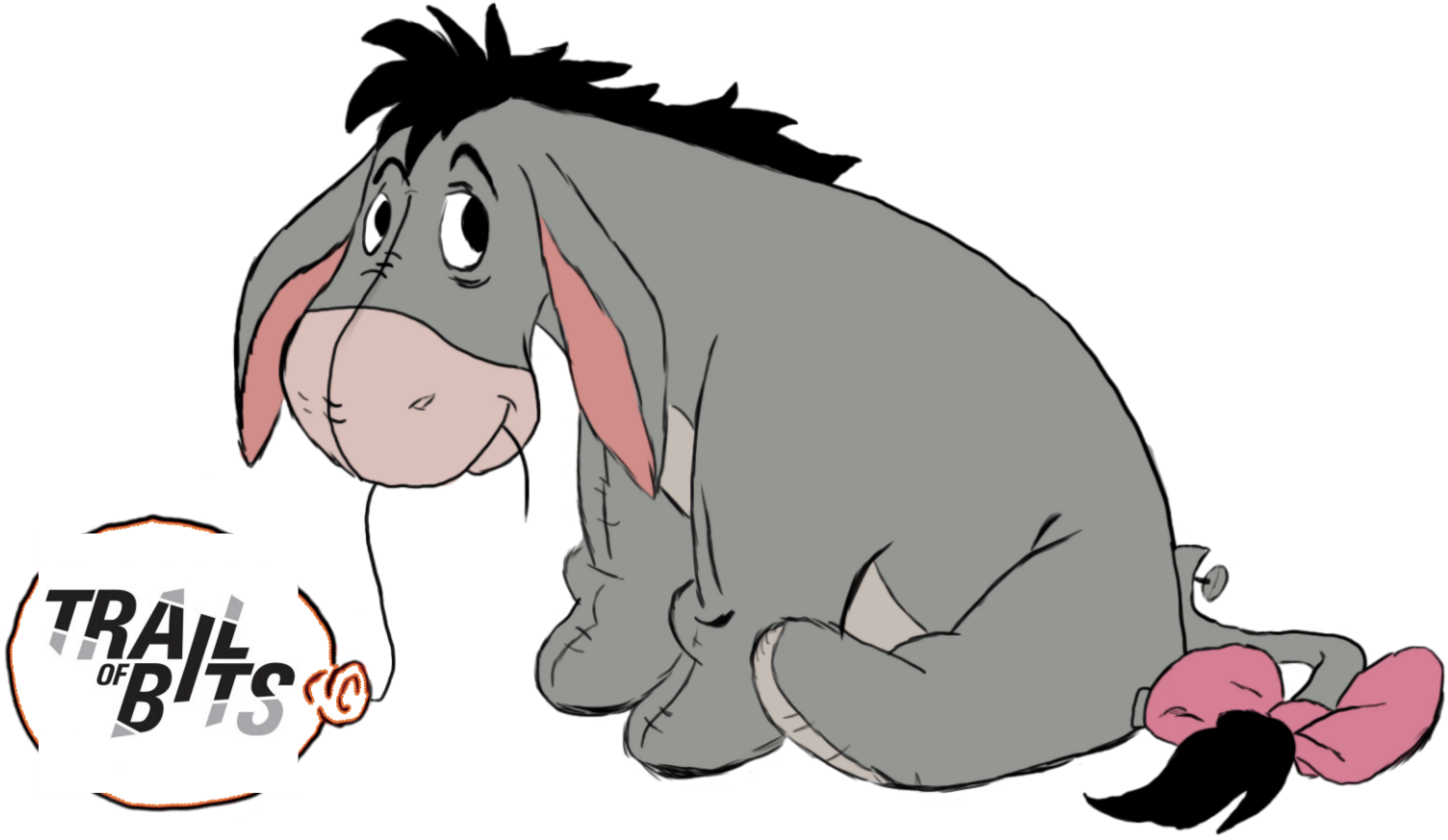
fuzzy failure

- what you fuzz matters
- but so does **how** you fuzz

i'm fuzzy on this one point

- as an aside, fuzzing, symbex, &c don't replace tests
  - tests show expected result, intent
  - fuzzing shows potential edge cases
  - symbex, absint show domain/codomain
- both are useful

fancy crypto won't save you



crypto in 2016

- mostly FIPS-140 types of stuff
- sometimes a rec for removing SHA
  - use PBKDF2, bcrypt, scrypt
- pining to use SRP

crypto in 2019

- work with actual cryptographers
- lots and lots of math
- better systems
  - libsodium, tink
  - ~~SRP~~ various PAKEs
  - Argon2
  - Oblivious Functions
- so everything is better... right?

crypto in 2019

- keys written to disk with `0777` or `0655`
- keys in memory that an attacker can access
- keys stored on GitHub
- lots of incorrect applications
    - signature? hash? who knows!
- must. use. all. the. keys.

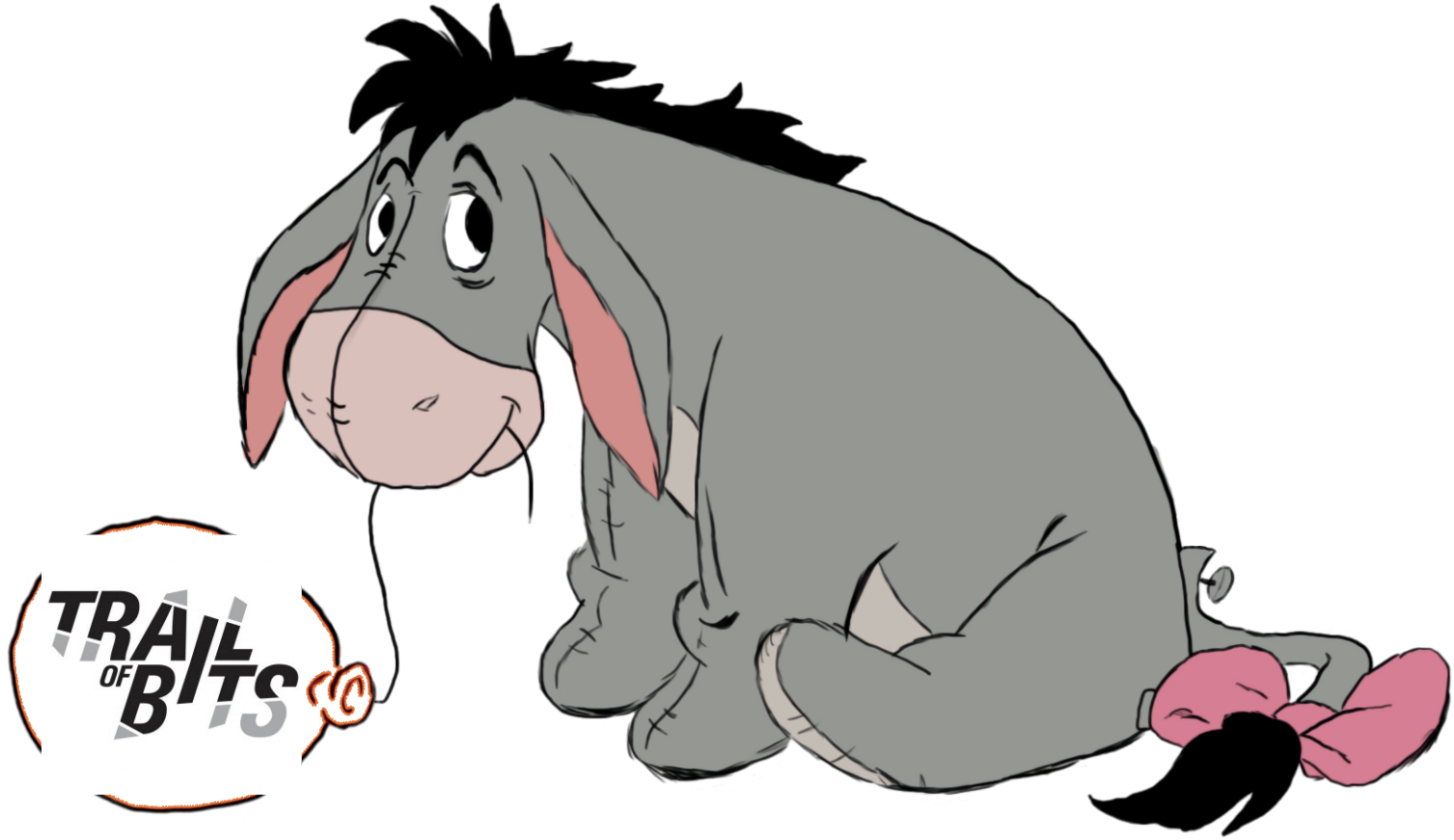
also, stop implementing your own crypto

- I don't want to review it
- it's probably wrong

ALSO if it's REALLY new, don't use it

- you may not implement it correctly
- the library isn't audited
- we don't understand it
- zk-SNARKs

operating systems won't save you



OSs in 2016

- mostly on prem
- Windows, Trusted Solaris, Linux
- some SELinux, some CAC access
- occassional orchestration, CM

OSs in 2019

- lots of orchestration
- almost 100% Linux
- containers == more security... right?

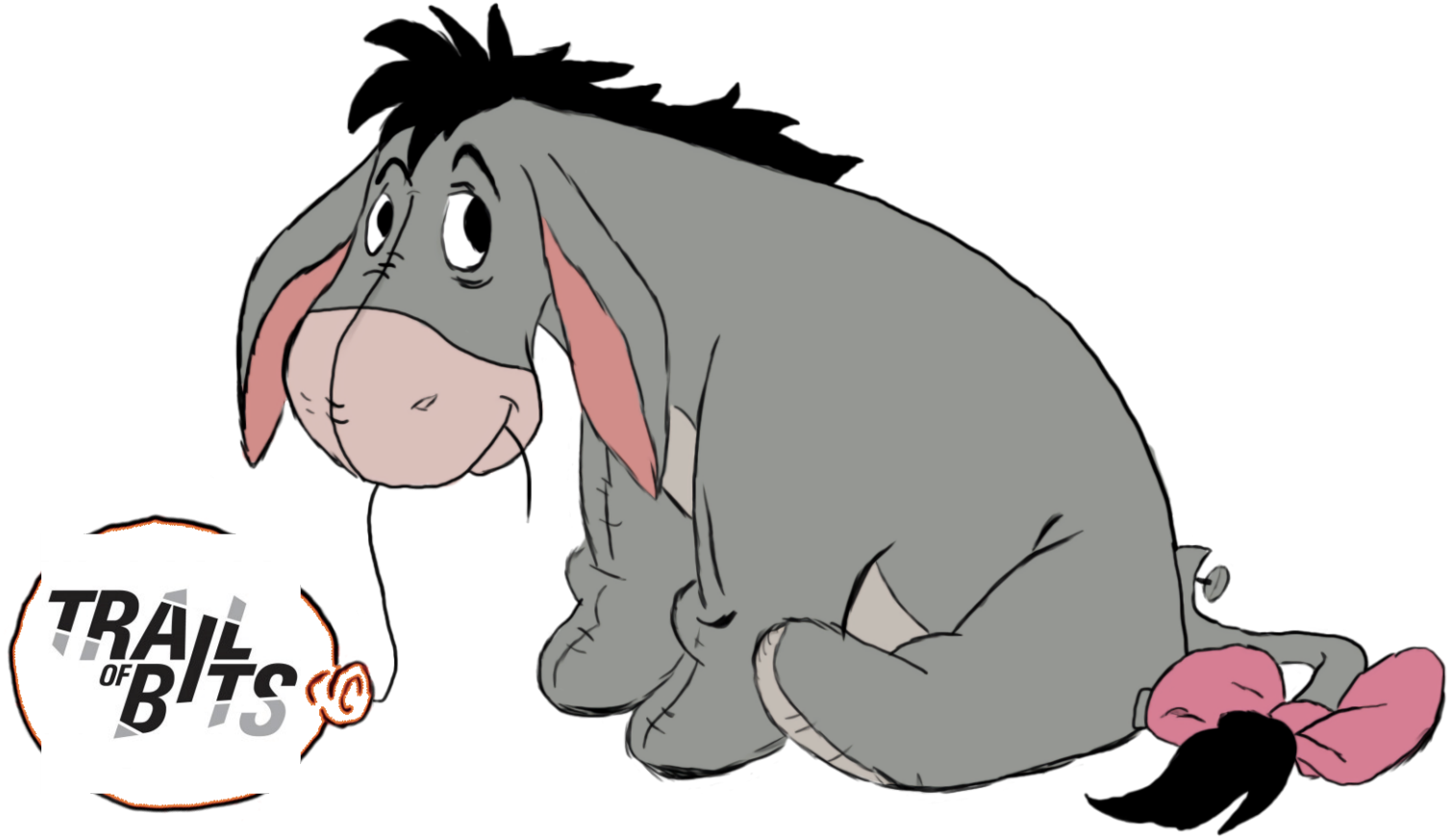
OSs in 2019

- ignoring simple docker failures
- k8s: by default disabled seccomp
  - since fixed
- docker: multiple vulns, often used w/o a KSM
- gvisor: disabled seccomp

OSs in 2019

- must pay attention to defaults
- by themselves, not a security mechanism
- not worse, but false sense of security

even if you get those things right, you don't win



you still don't win

- so you did everything right
  - want a cookie?
- there are still a number of otherways to lose
  - formally verify people?
  - crypto your way around a hardware bug?
  - SGX failures, k8s issues, &c.

the only winning move, is not to play

- reduce your surface area
- employ full range of:
  - threat modeling
  - maturity modeling
  - traditional SAST/DAST
  - fuzzing
  - design spec
  - verification (appropos of your risk)

and then you still die

- someone commits prod keys to GH

thanks!