Model-based Intrusion Detection System (IDS) for Smart Meters

Karthik Pattabiraman and Farid Tabrizi
Dependable Systems Lab
University of British Columbia (UBC)

My Research

- **Building fault-tolerant and secure software systems**
- **Application-level fault tolerance**
  – Software resilience techniques [DSN’14][DSN’13][DSN’12]
  – Web applications’ reliability [ICSE’14][ICSE’14][ESEM’13]
- **This talk**
  – Smart meter security [HASE’14][WRAITS’12]

Smart Meter Security

- **Smart meter Attacks**
  - No need for physical presence
  - Hard to detect by inspection or testing
  - Attacks can be large-scale

Analog Meter

Smart Meter

Security is a concern

Security is a concern

Topic: Security

Follow via:

Smart meter hacking tool released

Summary: *Termineter, an open-source tool designed to assess the security of smart meters, has been released.*

By Emil Protalinski for Zero Day | July 22,

Follow @emilprotalinski

(15:37 PDT)

Comments 0 Votes 4 Like 39

SecureState, an information security firm, on announced the public release of Termineter, framework written in Python that allows us security of Smart Meter utility meters over

09 APR 12

FBI: Smart Meter Hacks Likely to Spread

A series of hacks perpetrated against so-called "smart meter" installations over the past several years may have cost a single U.S. electric utility hundreds of millions of dollars annually, the **FBI** said in a cyber intelligence bulletin obtained by KrebsOnSecurity. The law enforcement agency said this is the first known report of criminals compromising the hi-tech meters, and that it expects this type of fraud to spread across the country as more utilities deploy smart grid technology.

Smart meters are intended to improve efficiency, reliability, and allow the electric utility to charge different rates for

FEDERAL BUREAU OF INVESTIGATION
INTELLIGENCE BULLETIN
Cyber Intelligence Section

27 May 2010

Goal

- **Goal:** Make smart meters secure
  - Build a host-based intrusion detection system (IDS)
  - Detect attacks early and stop them

- **Why is this a new challenge?**
  - Smart meters have unique constraints that make them different from other computing devices
  - Existing techniques do not offer comprehensive protection

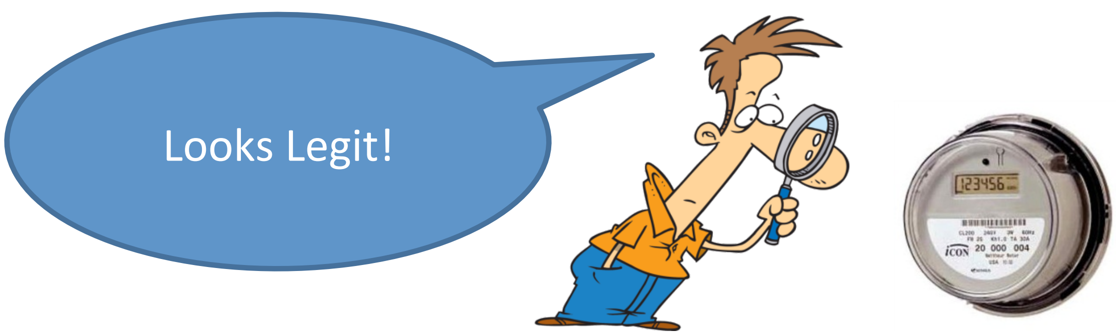
Outline

- Motivation and Goal
- **Prior work and constraints**
- Our approach
- Evaluation
- Formal modeling
- Conclusion

Prior Work on Smart Meter Security

- Network-based IDS [Barbosa-10][Berthier-11]

- Remote Attestation [LeMay-09][OMAP-11]



Why (bother with) Host-based IDS ?

- **Defense in depth**
  – Complement network-based IDS: False negatives
  – Can detect both physical and network attacks

- Remote attestation techniques do not cover attacks that change dynamic execution of the meter at runtime, e.g., control-flow hijacking

Constraints of smart meters

- **Performance**
  - Low-cost embedded devices; memory constrained
- **No false positives**
  - False-positive rate of 1% => 10,000 FPs in 1 million meters
- **Software modification**
  - Software has real-time constraints; no modifications
- **Low cost**
  - Rules out special cryptographic hardware or other additions
- **Coverage of unknown attacks**
  - Attacks are rapidly being discovered; zero-day attacks

Prior Work on Host-based IDS

System	Performance	No False Positives	No Software Modification	Low Cost	Unknown attacks
Dyck		X			X
NDPDA		X		X	X
HMM/NN/SVM	X		X	X	X
Statistical Techniques	X		X	X	X

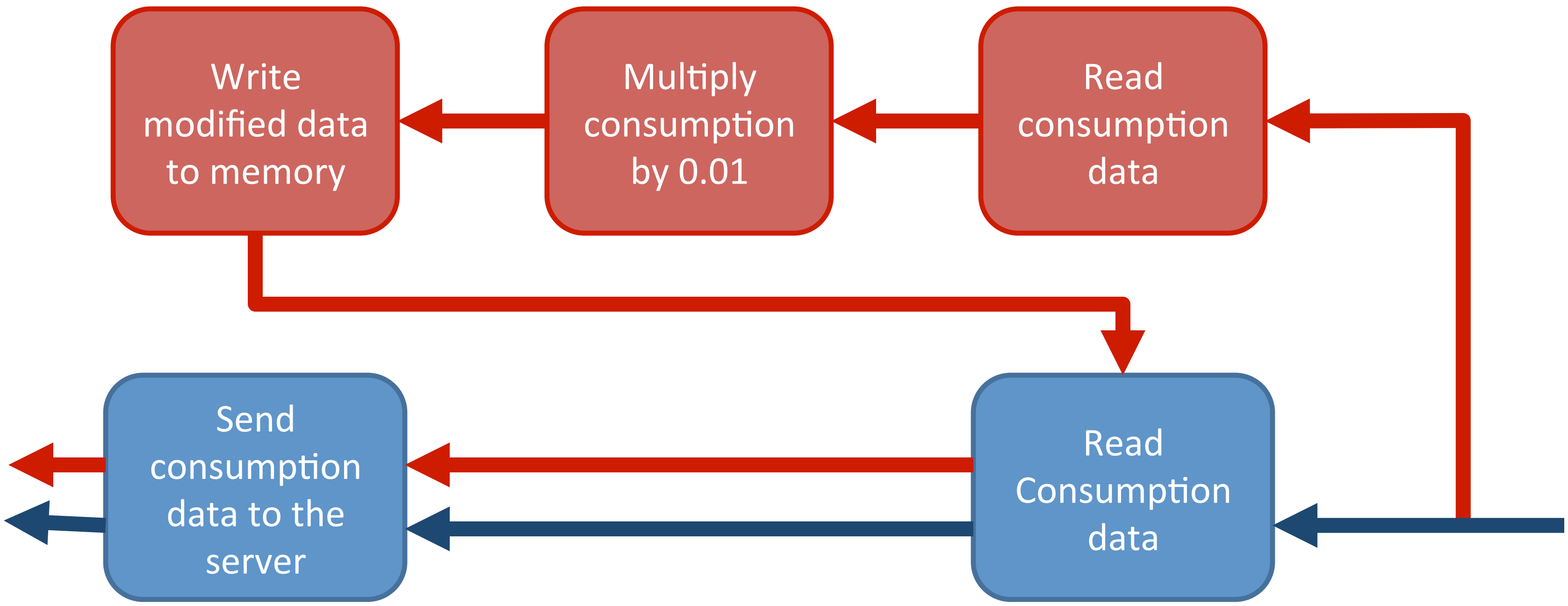
No existing host-based IDS can satisfy all five constraints: Need for new IDS

Outline

- Motivation and Goal
- Prior work and constraints
- **Our approach**
- Evaluation
- Formal modeling
- Conclusion

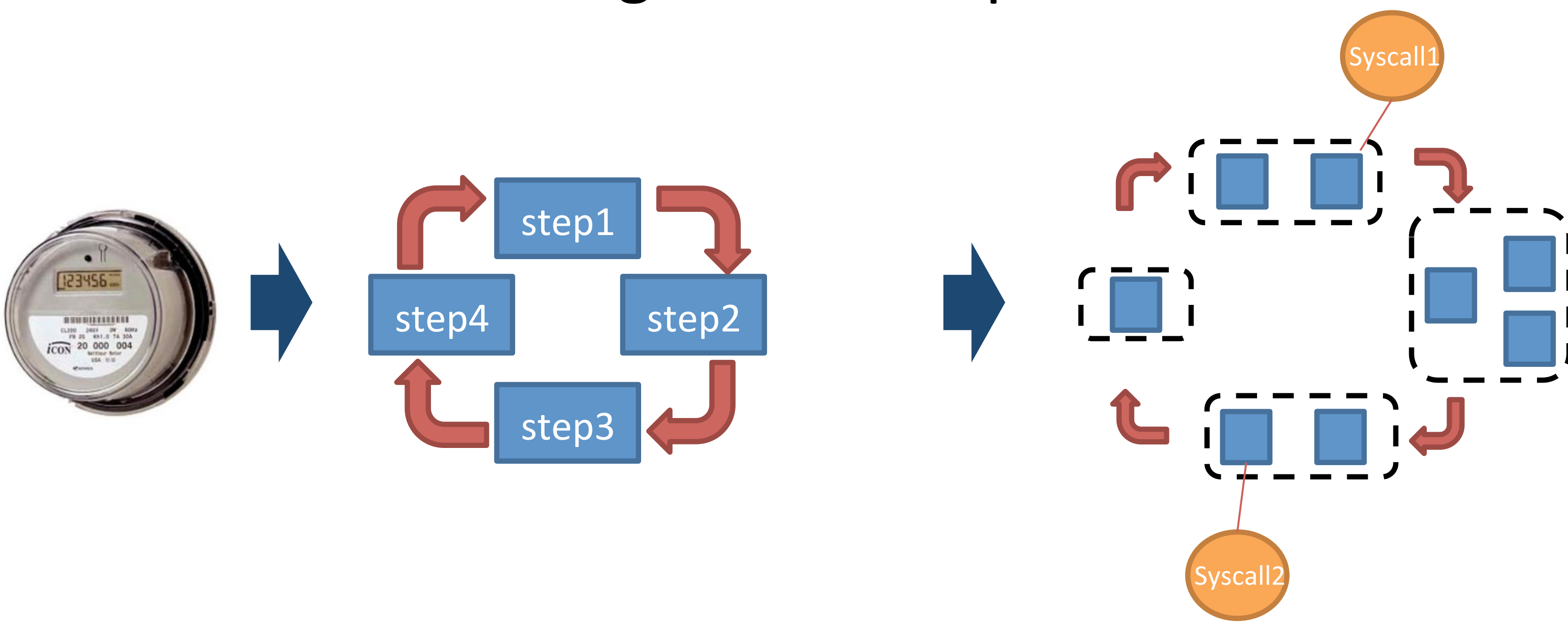
Threat model

- Adversary: wants to change the execution path of the software (in subtle ways)



Approach

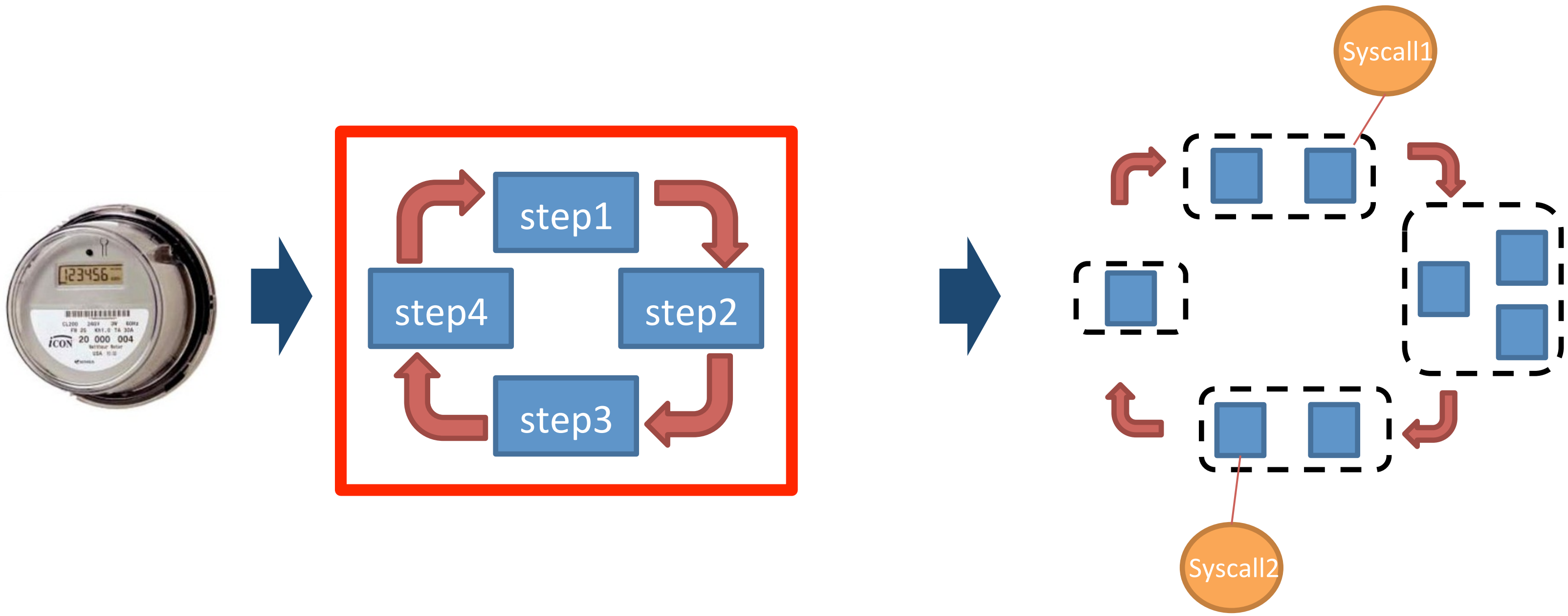
- Build a model of the meter software
  - Meters are designed to do specific tasks



Abstract Model

Concrete Model

Approach

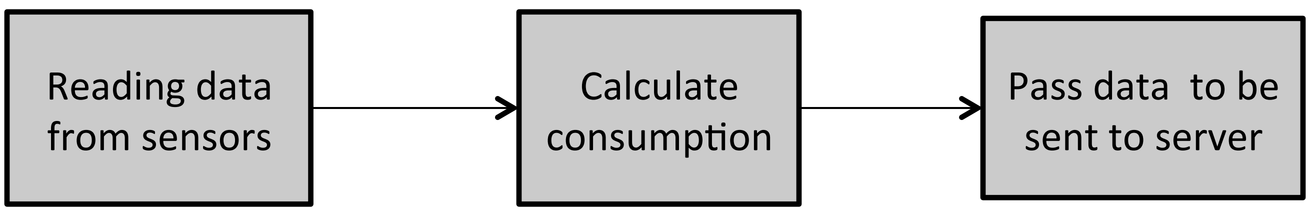


Abstract Model

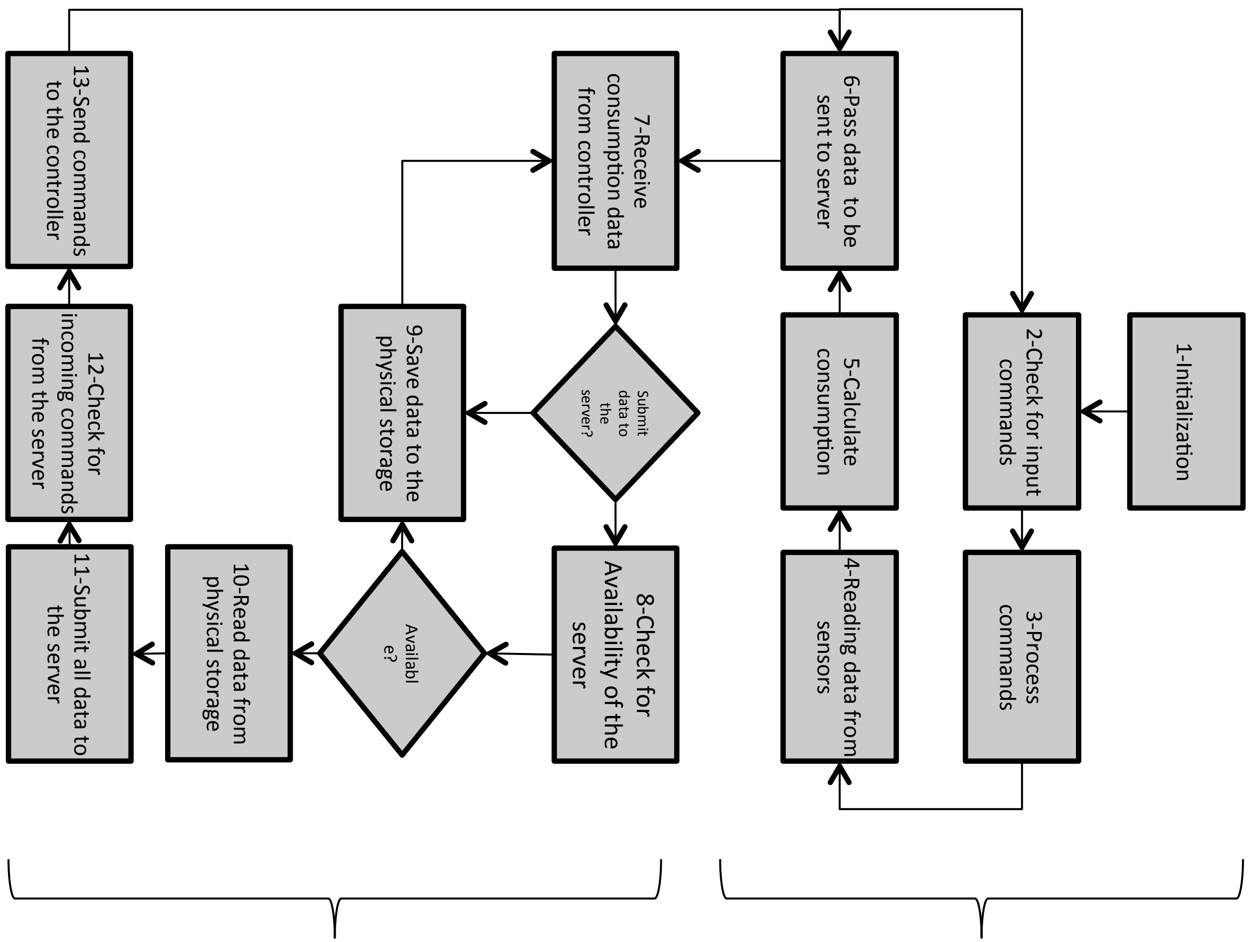
Concrete Model

Abstract Model

- Build an abstract model based on standard specifications of smart meter functionality

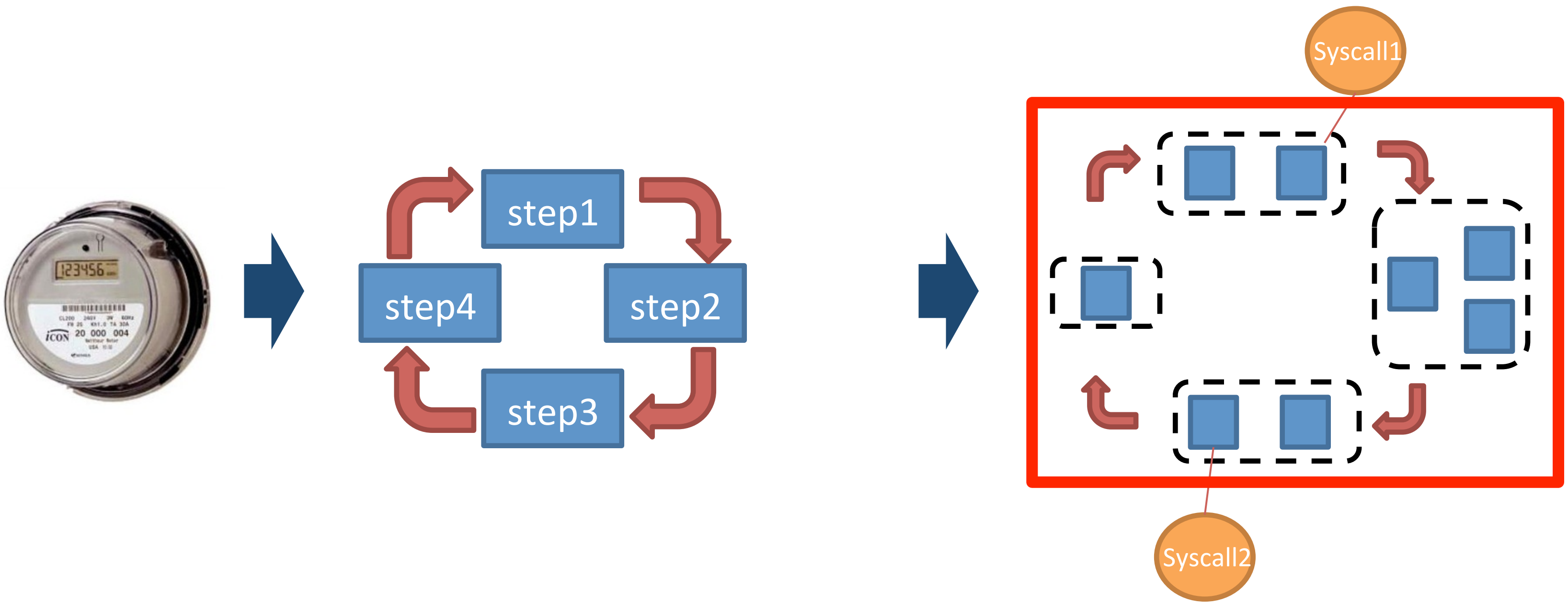


Abstract Model



Communication Processes **Controller Processes**

Approach



Abstract Model

Concrete Model

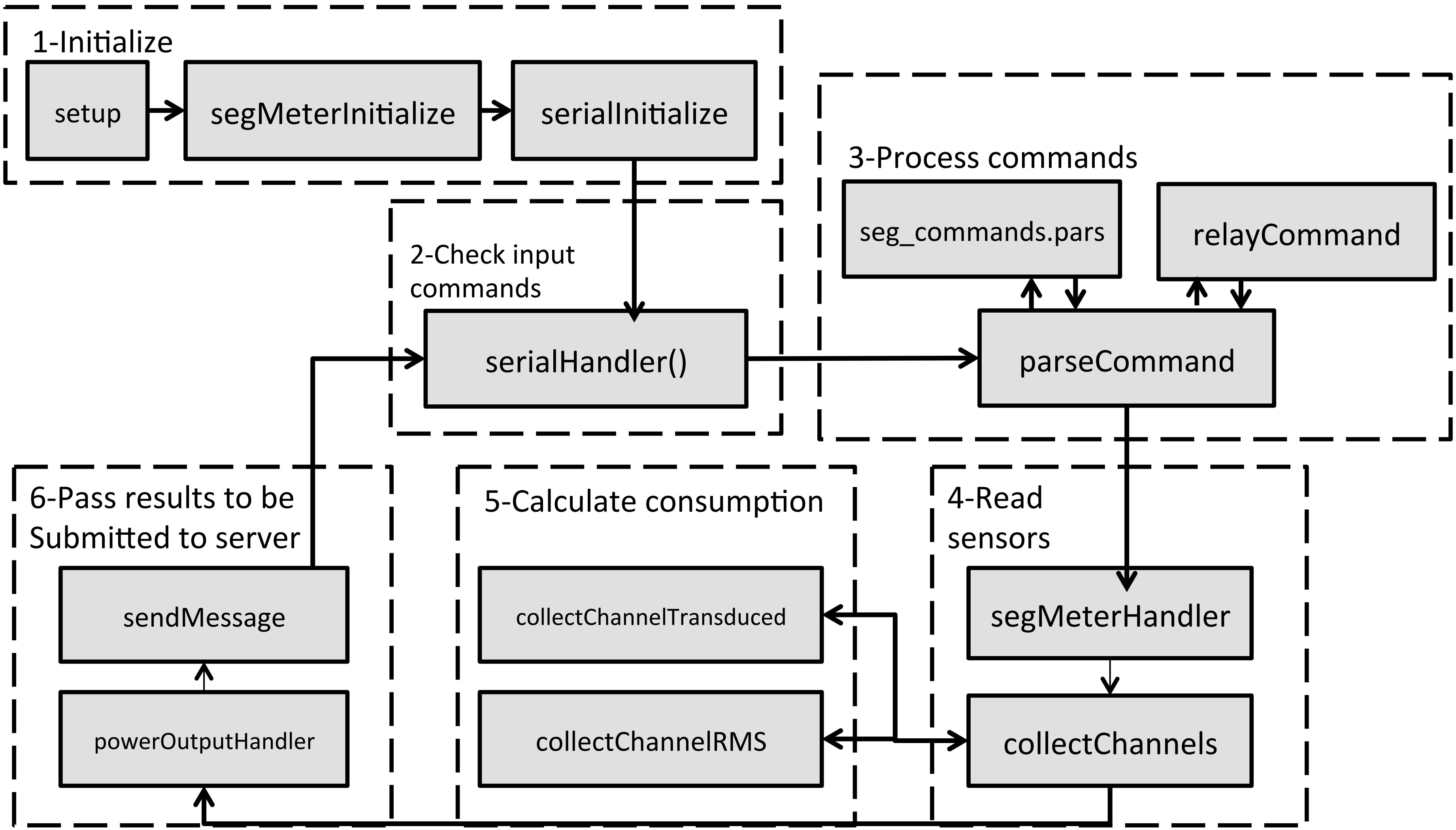
Building the concrete model

- **Use a tagging system**

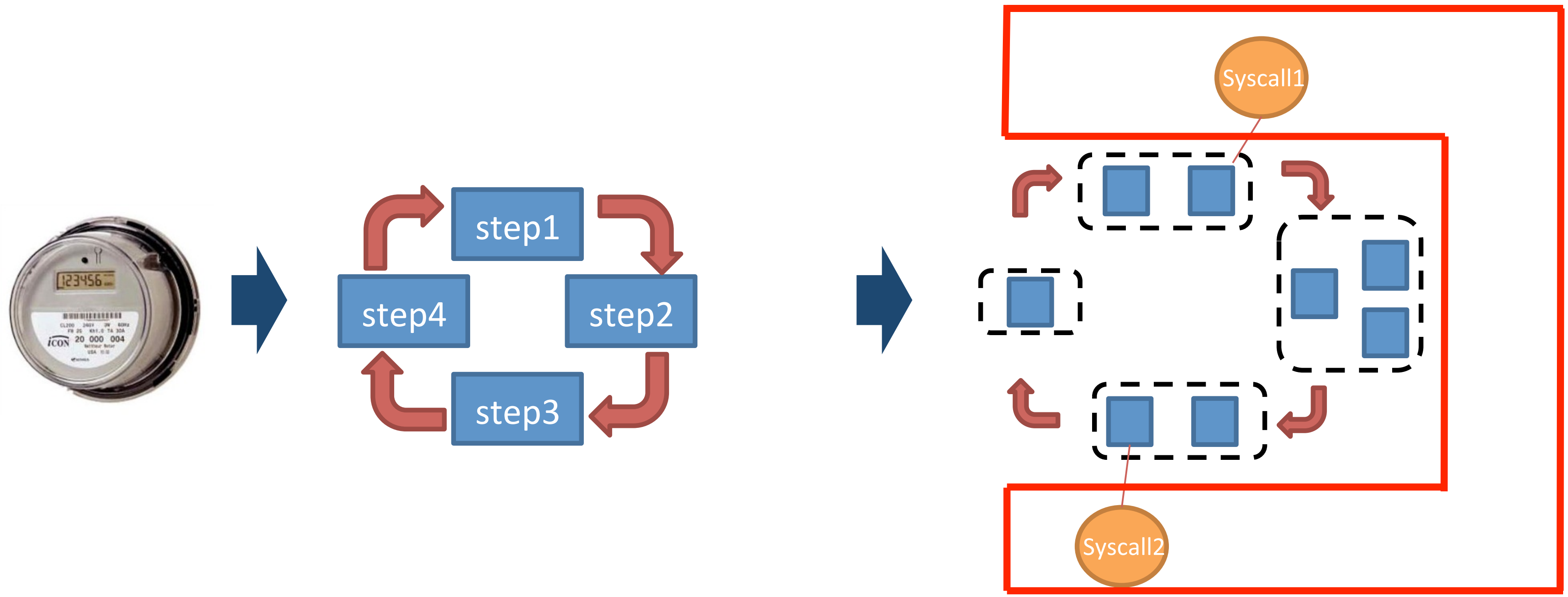
```
// <network, serial, b2>
SerialHandler()
{
...
}
```

- **Features**
  - Ease of use
  - Flexibility

Concrete Model



Approach

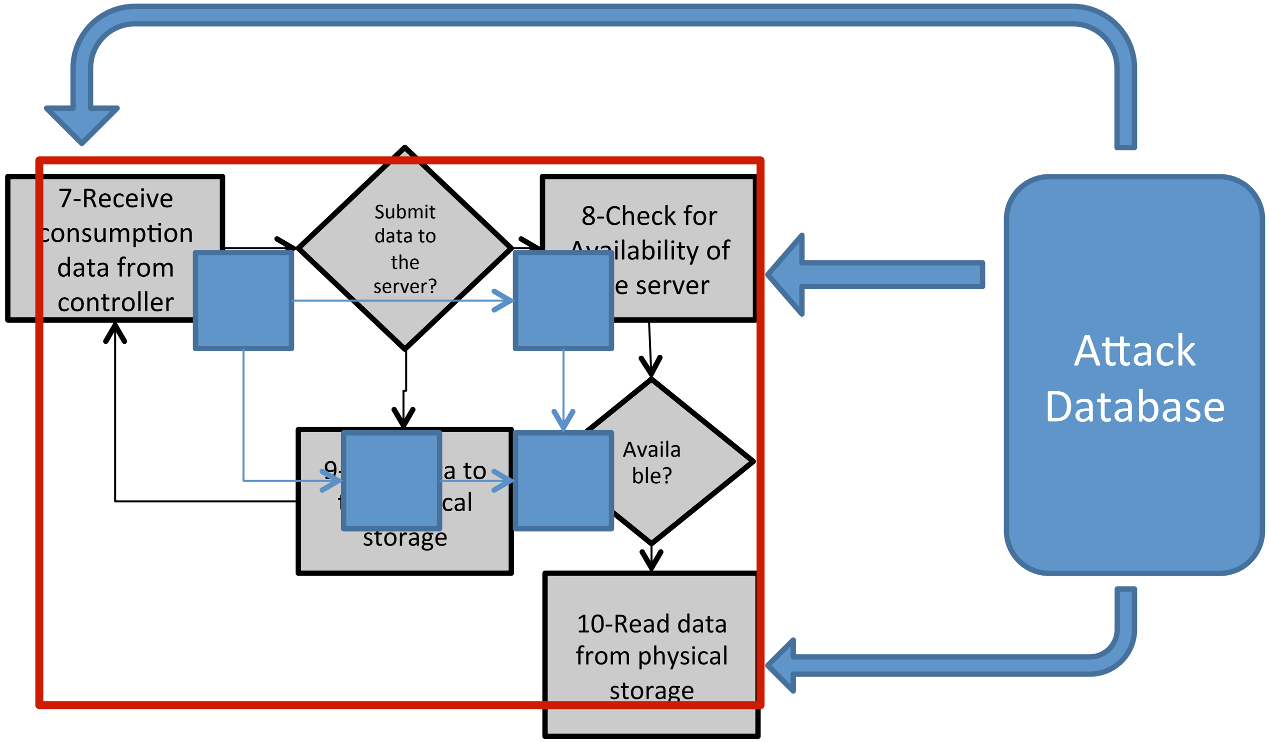


Abstract Model

Concrete Model

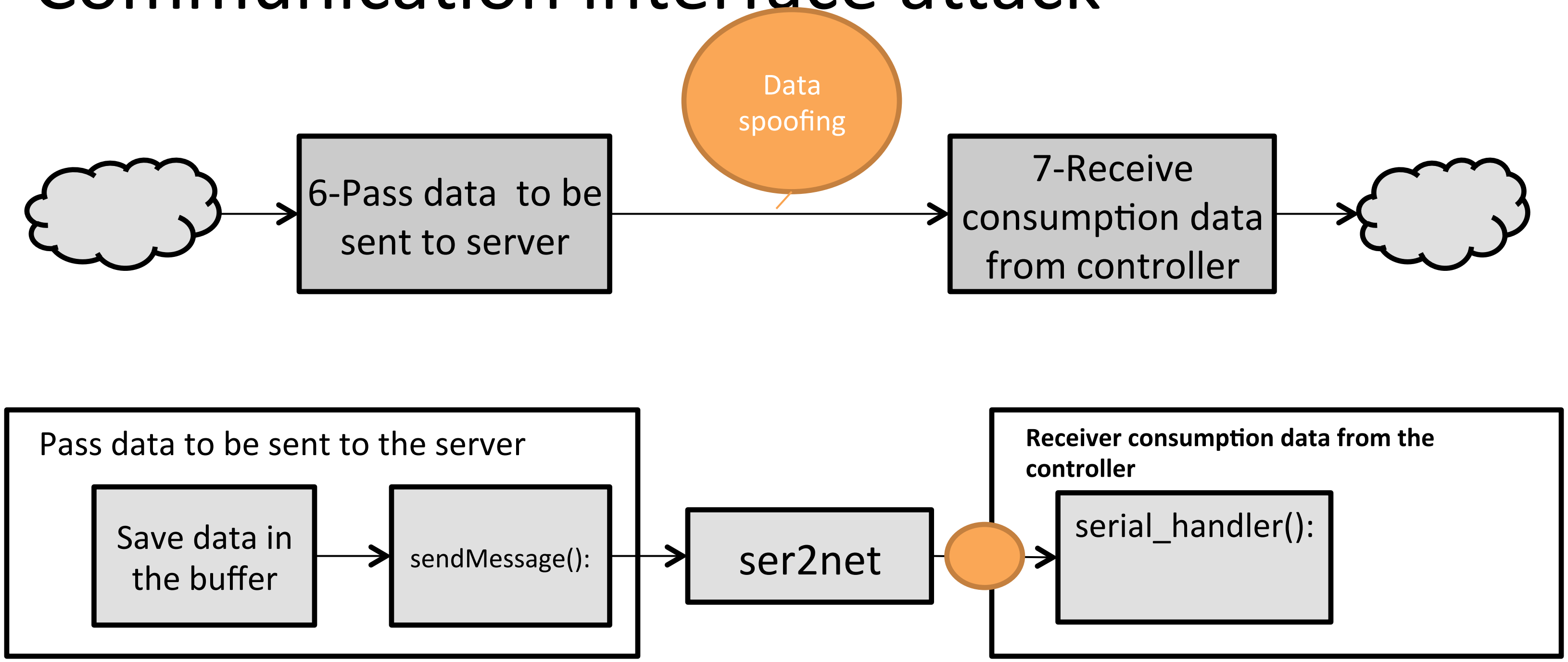
IDS Generation: Attack Database

- Build the IDS based on system calls



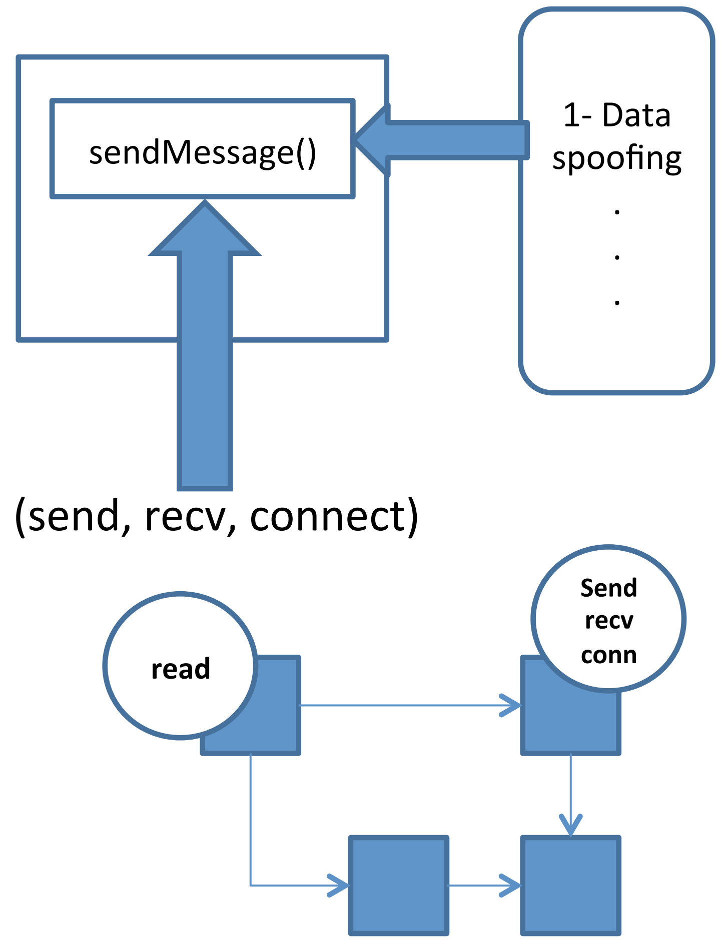
Example Attack

- Communication interface attack



System Call Selection: Algorithm

- Generate the set of all system calls of the meter
- Traverse the attack database
- Map the attacks to functionalities of the concrete model
- Map system calls to functionalities
- In the end: system calls associated with the attacks are mapped to the concrete model blocks
- Pick system calls that cover the most blocks until all blocks are covered
- Generate the state machine of the system calls based on the graph



Model-Based IDS: Implementation

- **Compile time:** Extract state machine of sys calls
  - Input: Annotated code
  - Output: state machine

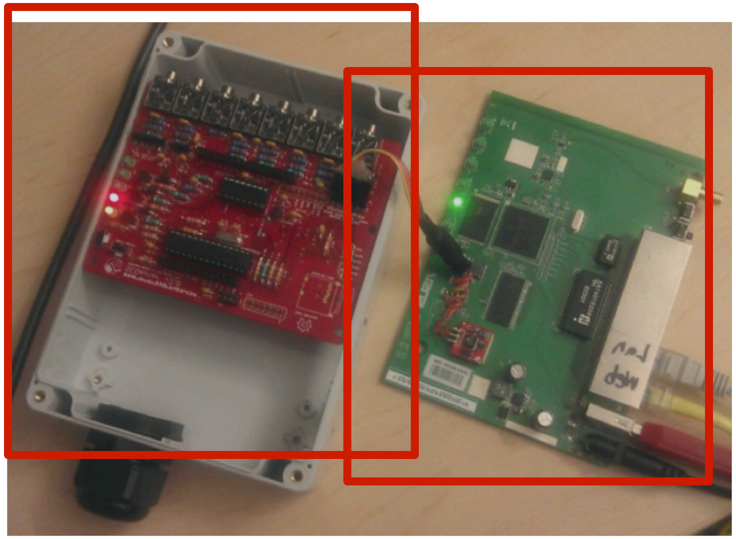
- **Run time:** Check sys call sequences
  - Logger: attaches *strace* to the process being monitored and logs system call traces
  - Checker: Runs every *T* second, parses the generated system calls, compares the logged trace with model

Outline

- Motivation and Goal
- Prior work and constraints
- Our approach
- **Evaluation**
- Formal modeling
- Conclusion

Experimental Setup

- SEGMeter
  - Arduino board
    - ATMEGA 32x series
    - Sensors
  - Gateway board
    - Broadcom BCM 3302 240MHz
    - 16 MB RAM
    - OpenWRT Linux
  - IDS runs on Gateway board



Results: Performance

- **Performance**
  - Tme taken to check the syscall trace / time taken to execute the meter software - produce the trace

Memory available	12 MB	9 MB	6 MB
Full-trace IDS	165.2%	214.6%	315.1%
Our Model-based IDS	4.0%	4.0%	4.0%

Full-trace IDS cannot keep up with the software, while our model-based IDS incurs low overheads

Results: Coverage (Known Attacks)

- **Detection (Known attacks)**
  - Implemented four different attacks [WRAITS'12]
    - Communication interface attack
    - Physical memory attack
    - Buffer filling attack
    - Data omission attack
  - **Our Model-Based IDS detects all four attacks**
    - If undetected, the attacks lead to severe consequences

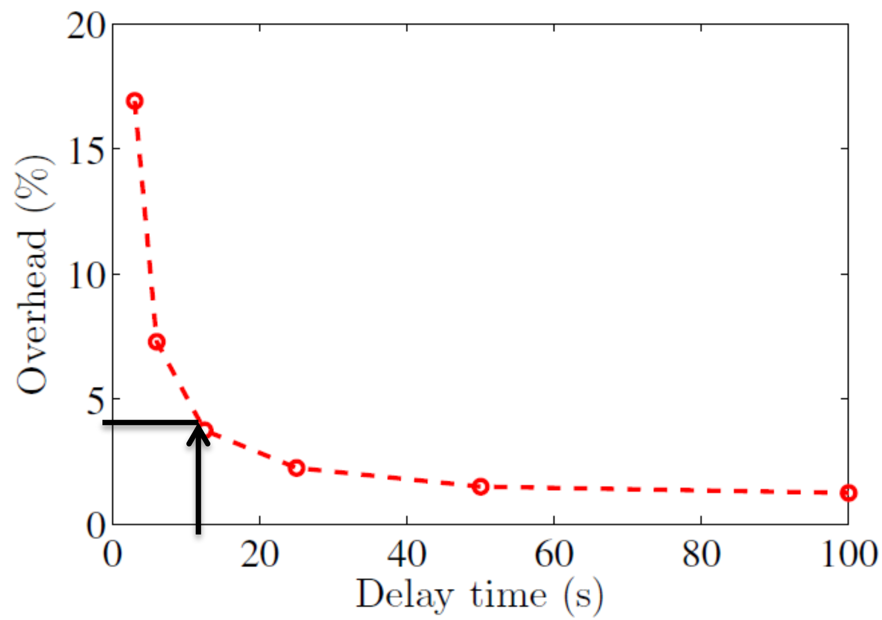
Results: Coverage (Unknown Attacks)

- **Detection (Unknown attacks)**
  - **Code injection**
    - Select a procedure to inject in the smart meter
    - Mutate the procedure by copying and pasting 1-8 lines of code from some other part of it (harder to detect)

Component	Random (%)	Popular system calls (%)	Full-trace (%)	Model-based		
				Minimum	Average	Maximum
Server communication	32	36	92	59	62	63
Storage and retrieval	14	44	84	73	74	78
Serial communication	42	28	88	67	72	74
Averagel	29.3	36.0	88.0	67.4	69.6	71.7

Results: Monitoring Latency

- Monitoring latency
  - Smaller *T*: Faster detection, higher performance overhead
  - We pick *T*= 10s
    - Low performance overhead: 4%
    - Full trace can't keep up even with T=60s



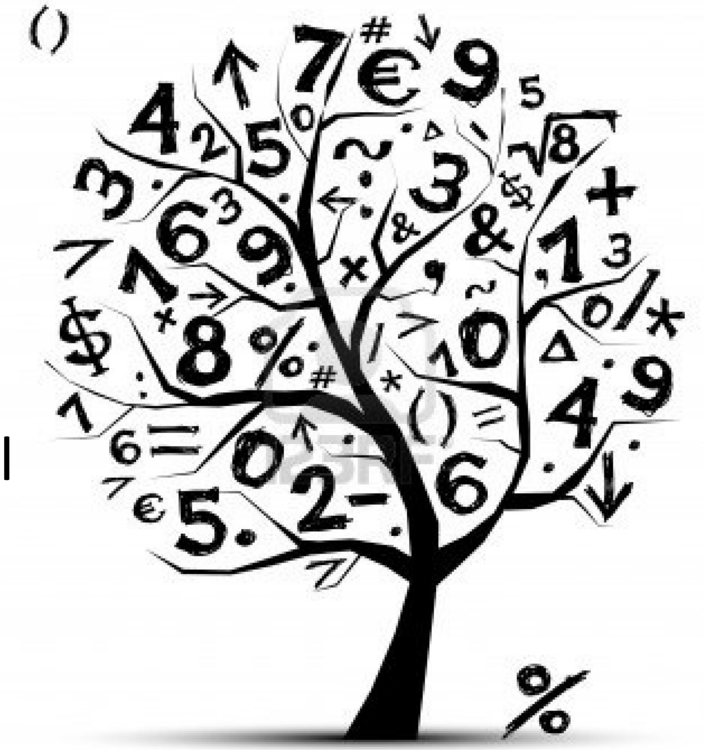
T = 10 s

Outline

- Motivation and Goal
- Prior work and constraints
- Our approach
- Evaluation
- **Formal modeling**

Towards formal modeling

- Manual checking of IDS
  - Inaccuracy
  - Effort
- Formal Modeling
  - Formal definition of the flaws
  - Formal definition of the model
- Goals: Speed and accuracy



Formal Modeling: Approach

- **We model the operations of the smart meter**
  - Low level (code level)

- **What do we do with the model?**
  - Define invariants:
    - Is it possible to change the consumption data?
    - Is it possible that data not be stored?
    - Is it possible to skip consumption calculation?

- **Test the model against the invariants**
  - Find the flaws → provide potential solutions

Formal Modeling Approach - 1

- **We model the operations of the smart meter**
  - Low level (code level)

```
function process_seg_response(response)

  local win = true
  local command = nil
        …
  if (response:sub(1, 7) == "(site= ") then
                …
  if (response:sub(1, 6) == "(node ") then
                …
  return win
```

Our input is the code

```
module process_resp(response, result)
{
   input response: string;
   output result: string;
   if (…)
      result = time  + consumption;
      ….
}
```

- **Use the variables of the code as input**
- **Rewrite the statements**

Formal Modeling Approach - 2

- **What do we do with the model?**
  - Define checks for different invariants

```
module process_resp(response, result)
{
  input response: string;
  output result: string;
  if (…)
    result = time  + consumption;
    ….
  cond1: assert ~(result == nil)
  cond2: assert (response → consumption > 0)
   …
}
```

Will be checked against all possible inputs

Formal Modeling Approach - 3

- **Test the IDS against the model and invariants**
  - Find the flaws → provide potential solutions

Example:
response == “” → consumption = 0 (default value)

Attacker can make the string empty (“”) even without knowing the encoding scheme

Solution

Add a check for empty string and raise an alarm for it

Outline

- Motivation and Goal
- Prior work and constraints
- Our approach
- Evaluation
- Formal modeling
- **Conclusion**

Conclusion

- **Smart meters have special constraints that make existing host-based IDSes impractical**
- **Our model-based IDS: practical for smart meters**
  - Low performance overhead
  - Good detection coverage
  - Low detection latency
- **Formal modeling can help automate the analysis of the software: provide strong guarantees**

Future Work and Discussion

- Extend to other SCADA systems (e.g., transportation systems, oil pipelines etc)
- Build a generic framework to reason about trading-off security for performance
- Automated inference of concrete model through static analysis without annotations