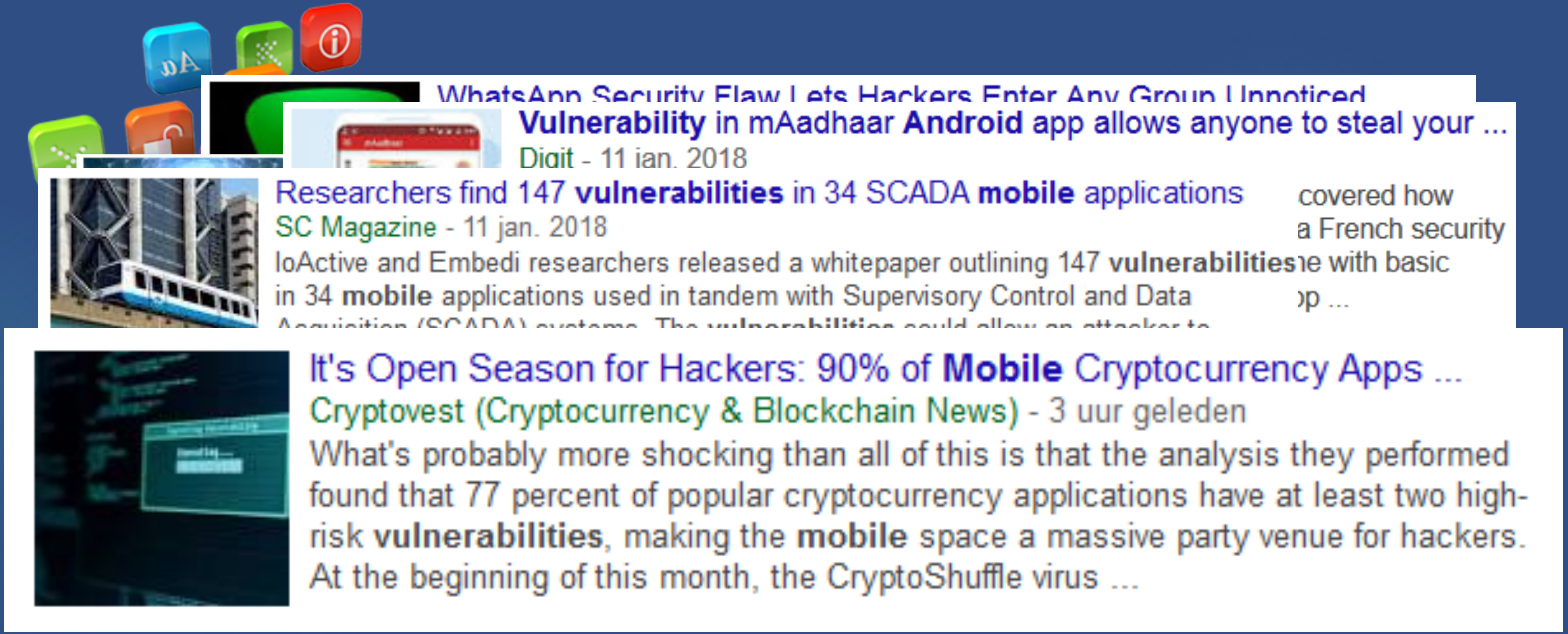
Vulnerability Detection in Mobile Applications using State Machine Modeling

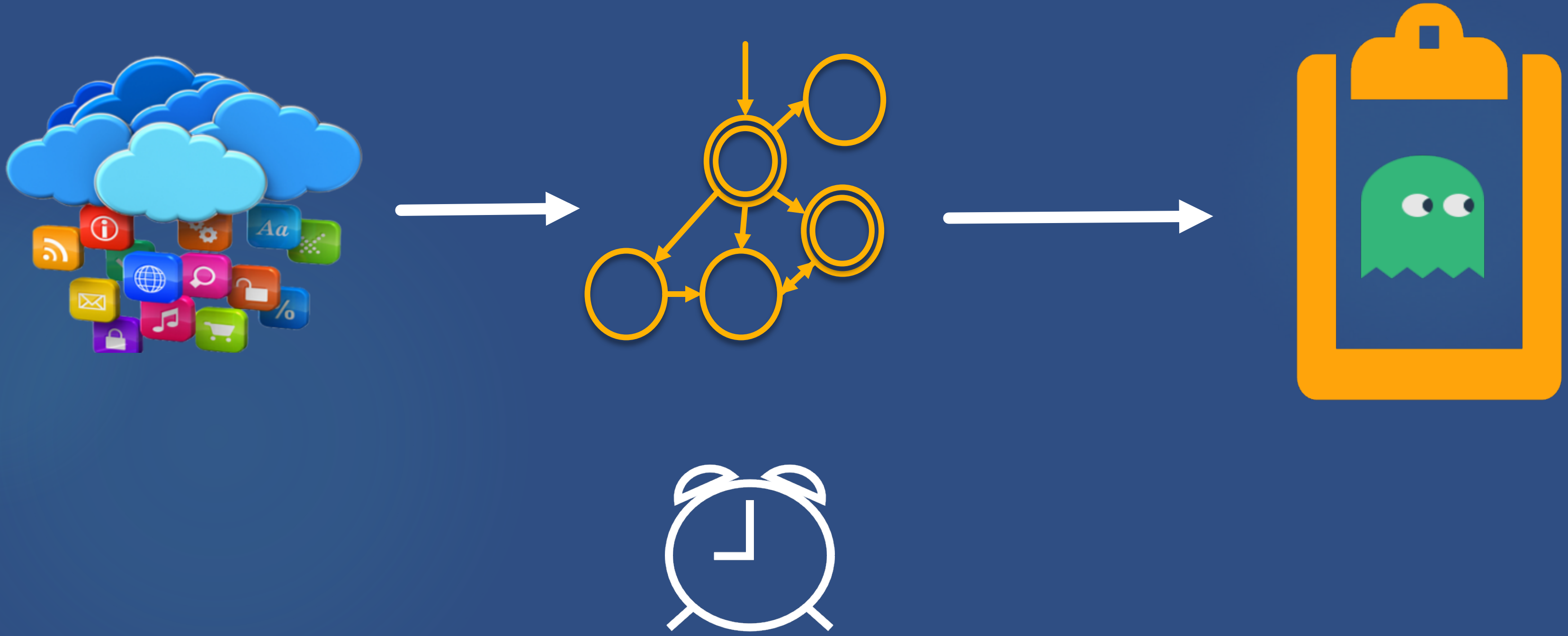
Wesley van der Lee
TUDelft
April 23th, 2018

Security Protocol Implementations: Development and Analysis (SPIDA)

Introduction



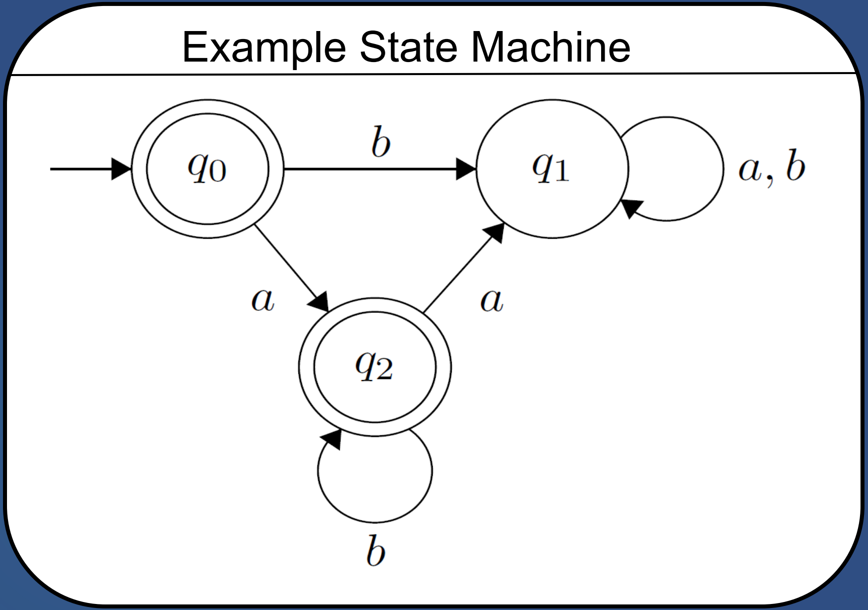
Research Goal



Outline

- State Machine Learning
- Mobile State Machine Learning
- Vulnerability Detection
- Results
- Conclusion

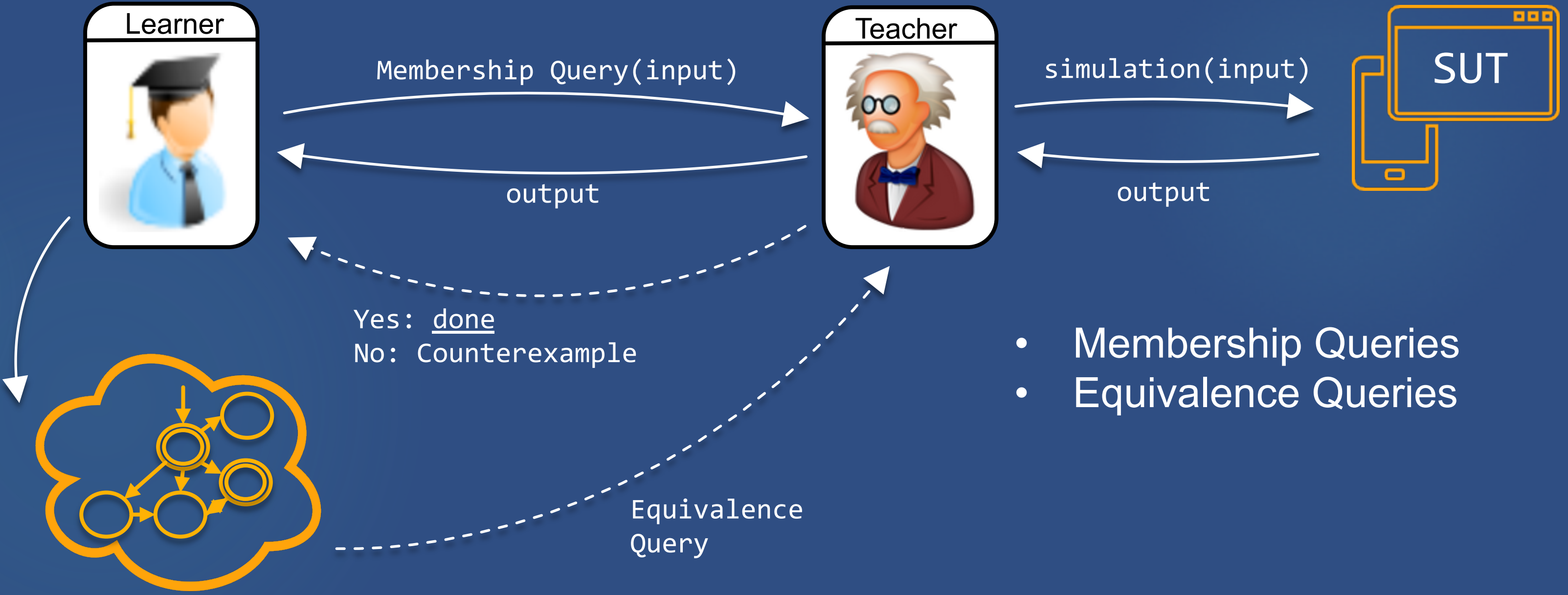
Definition of a State Machine



- States= $\{q_0, q_1, q_2\}$
- Start= q_0
- Transitions =
- Alphabet= $\{a,b\}$
- Accepting states= $\{q_0, q_2\}$

MAT Framework

- Minimally Adequate Teacher (*Angluin, 1988*)

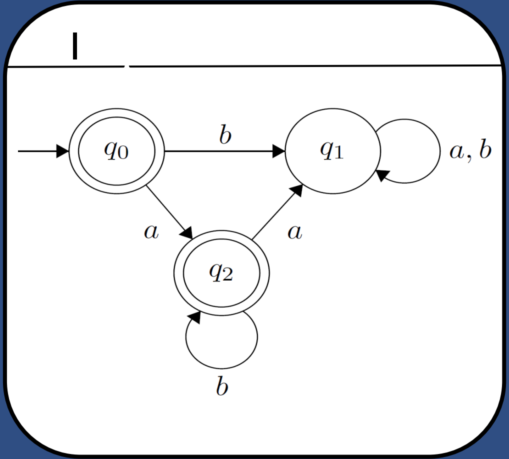


- Membership Queries
- Equivalence Queries

Angluin, Dana. "Queries and concept learning" *Machine learning* 2.4 (1988): 319-342.

Active Learning with L*

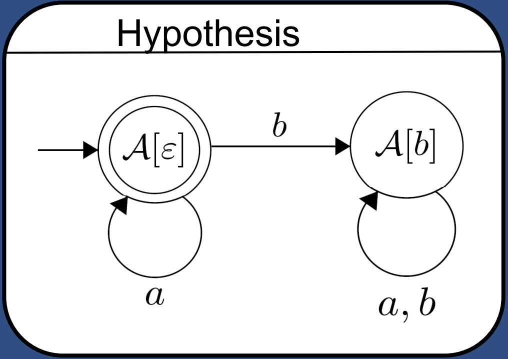
Membership Query {ε,a,b}
Output: {ε=1,a=1,b=0}



S = Prefixes (row)

E = Suffixes (column)

	ε
ε	1
b	0
a	1
ba	0
bb	0

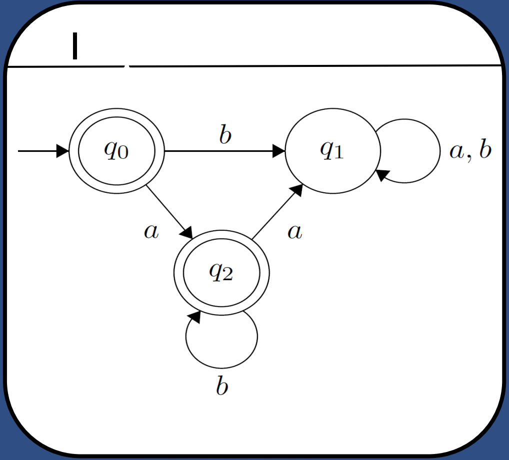


Alphabet={a,b}

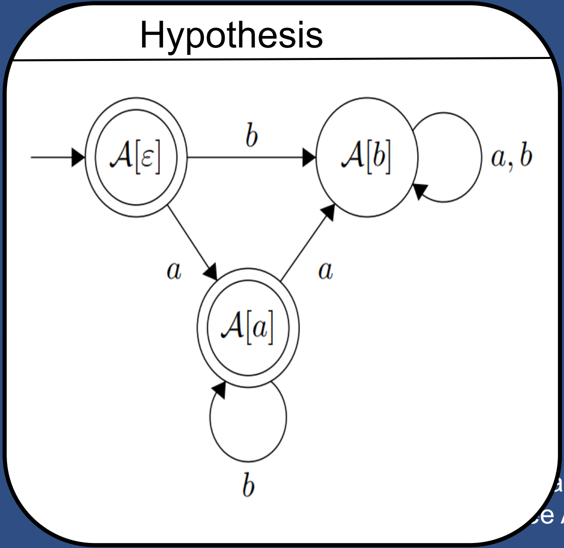
- Rows:
  - Unique rows: identify states

Active Learning with L*

NO ! Counterexample: ab
YES: Equivalent!
Equivalence Query H1 / H2

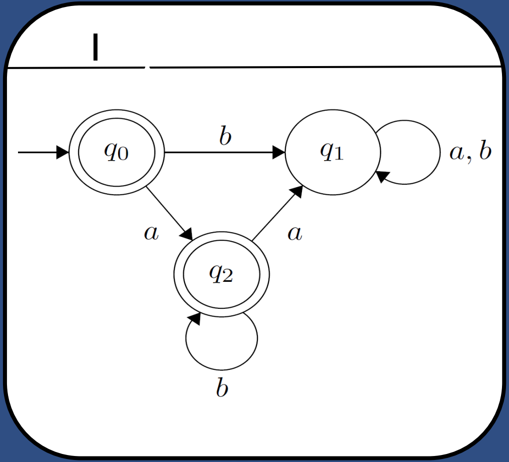


	ε	b
ε	1	0
b	0	
a	1	1
ba	0	
bb	0	
aa	0	
ab	1	1



…alk Howar, and Bernhard Steffen. "The TTT Algorithm: A …e Approach to Active Automata Learning." RV. 2014.

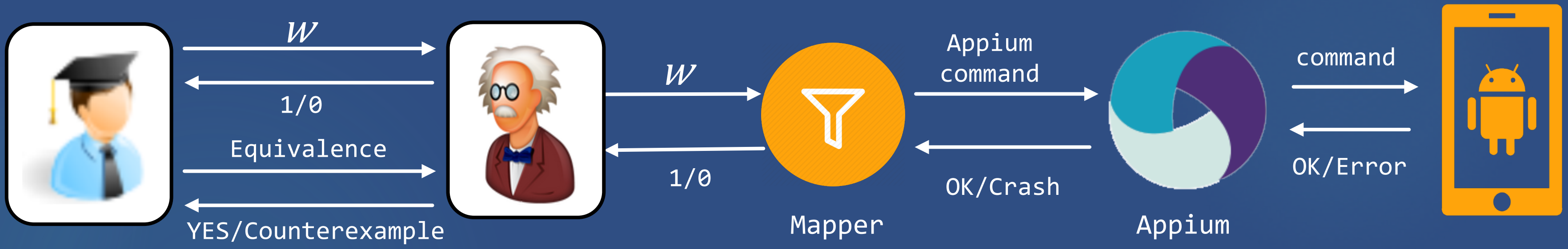
Equivalence Queries



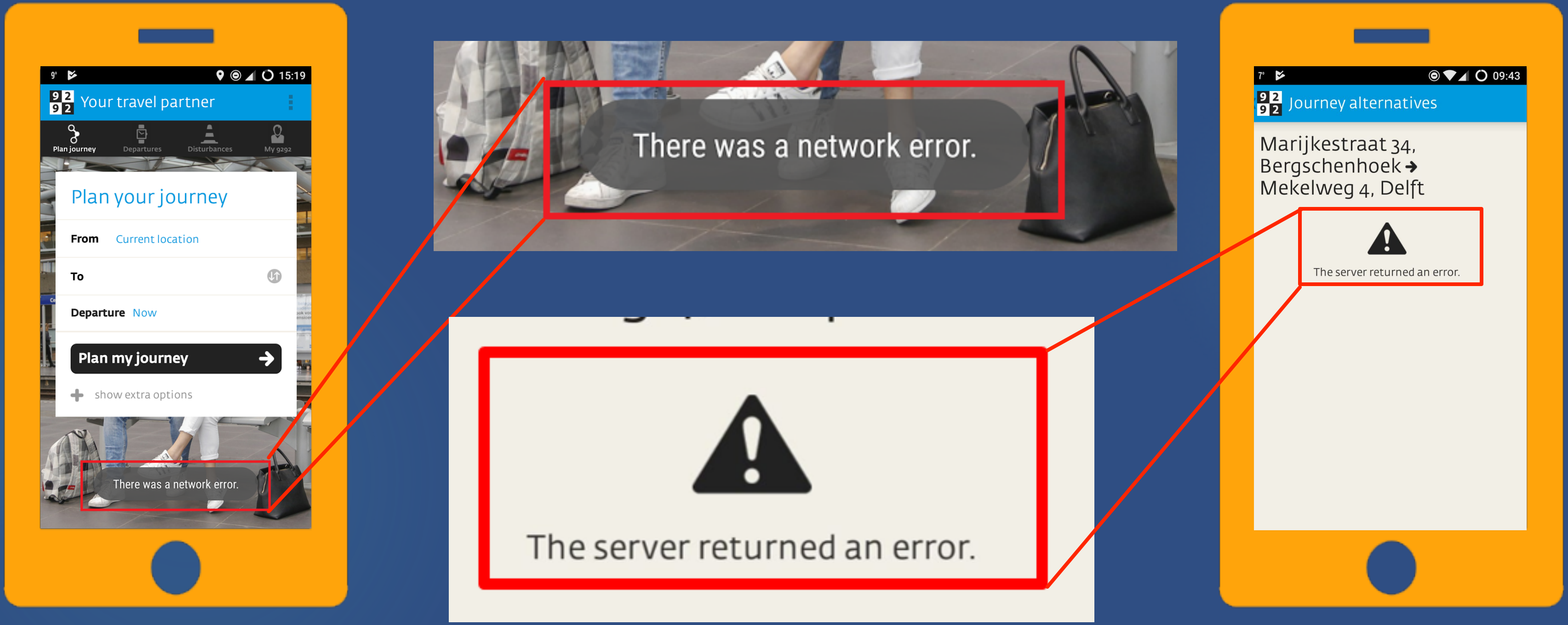
- Conformance Testing
  - Random Walk
    - HappyFlow
  - W-Method
    - $P \times \Sigma \uparrow n \times W$ (Chow, 1978)
    - $P \times \Sigma \uparrow n \times W'$ (Smetsers et al., 2016)

Chow, T.S. "Testing software design modeled by finite-state machines." IEEE transactions on software engineering 3 (1978): 178-187.
Smetsers, R. et al. "Minimal separating sequences for all pairs of states." International Conference on Language and Automata Theory and Applications. Springer International Publishing, 2016.

Mobile State Machine Learning

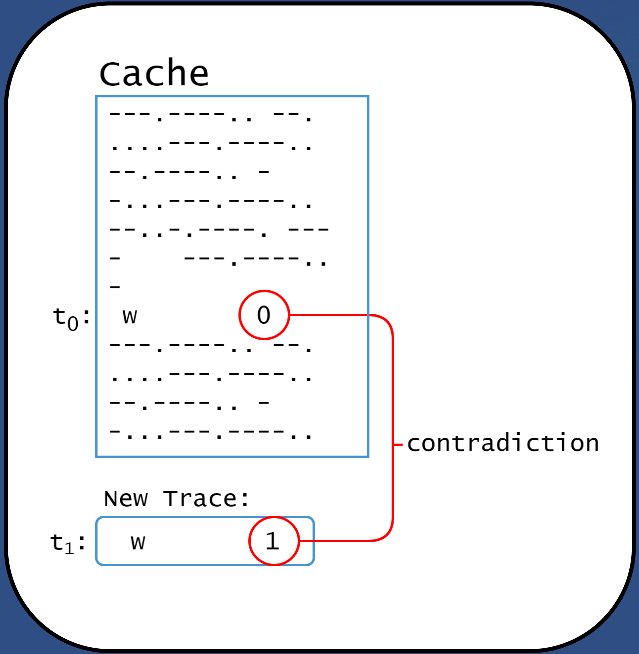


Problems when learning

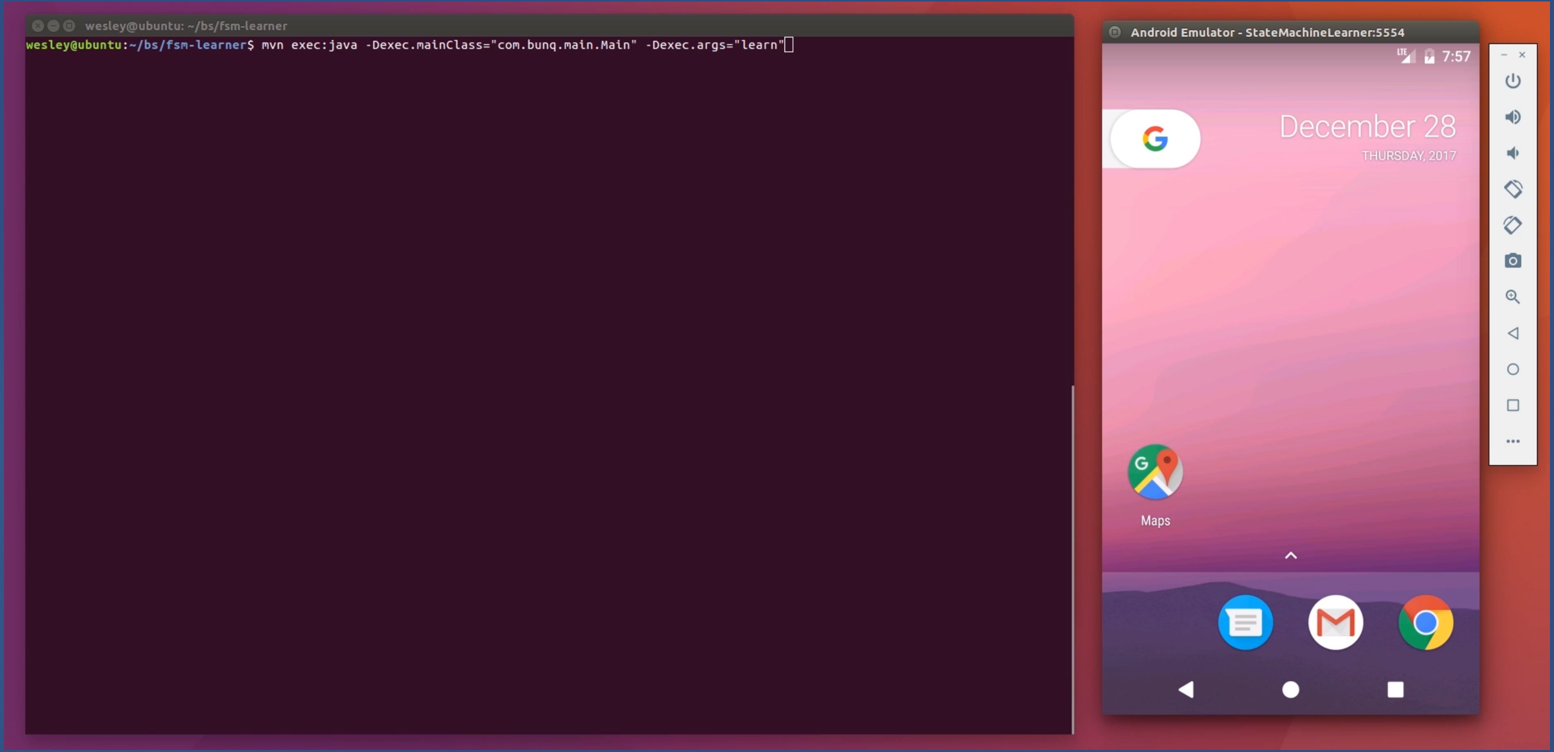


Problems when learning

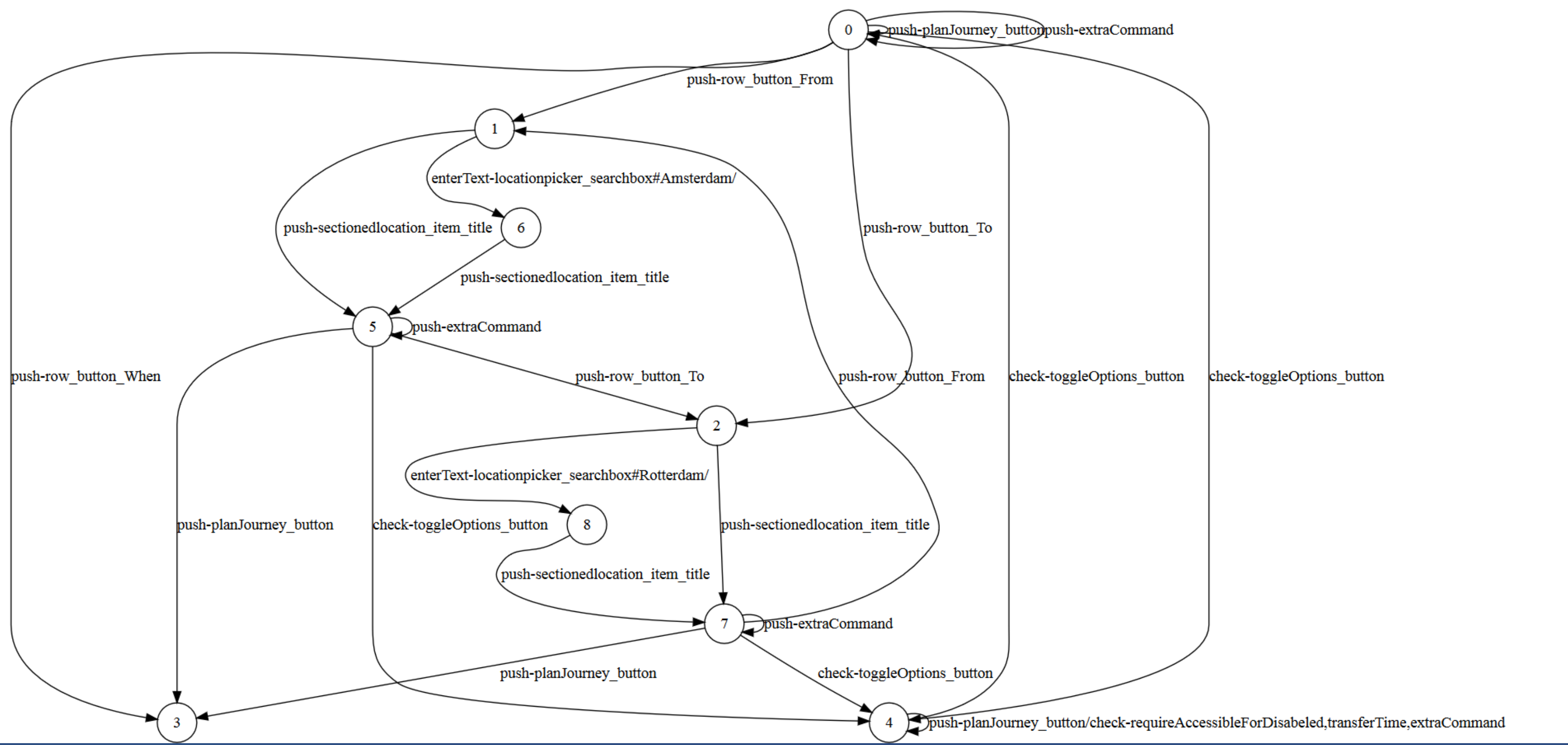
- Non-deterministic behavior
- Inconsistent cache
- Cache Roll-Back



Video


wesley@ubuntu: ~/bs/fsm-learner
wesley@ubuntu:~/bs/fsm-learner$ mvn exec:java -Dexec.mainClass="com.bunq.main.Main" -Dexec.args="learn"
Android Emulator - StateMachineLearner:5554
7:57
December 28
THURSDAY, 2017
Maps

9292 Model


0
push-planJourney_buttonpush-extraCommand
push-row_button_From
1
enterText-locationpicker_searchbox#Amsterdam/
push-sectionedlocation_item_title
6
push-row_button_To
push-sectionedlocation_item_title
5
push-extraCommand
push-row_button_When
push-row_button_To
push-row_button_From
check-toggleOptions_button
check-toggleOptions_button
2
enterText-locationpicker_searchbox#Rotterdam/
push-planJourney_button
check-toggleOptions_button
8
push-sectionedlocation_item_title
push-sectionedlocation_item_title
7
push-extraCommand
push-planJourney_button
check-toggleOptions_button
3
4
push-planJourney_button/check-requireAccessibleForDisabeled,transferTime,extraCommand

Vulnerability Detection

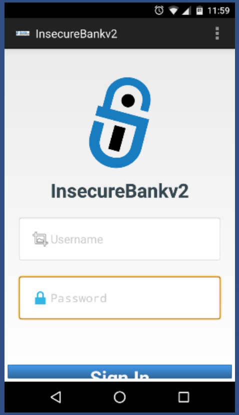
- Enrich Models
- Define Vulnerability Algorithms
  - OWASP Top10 Mobile
    - i.e. Insecure Authentication:

      Does there exist a path to a node after the login state without traversing the login state?

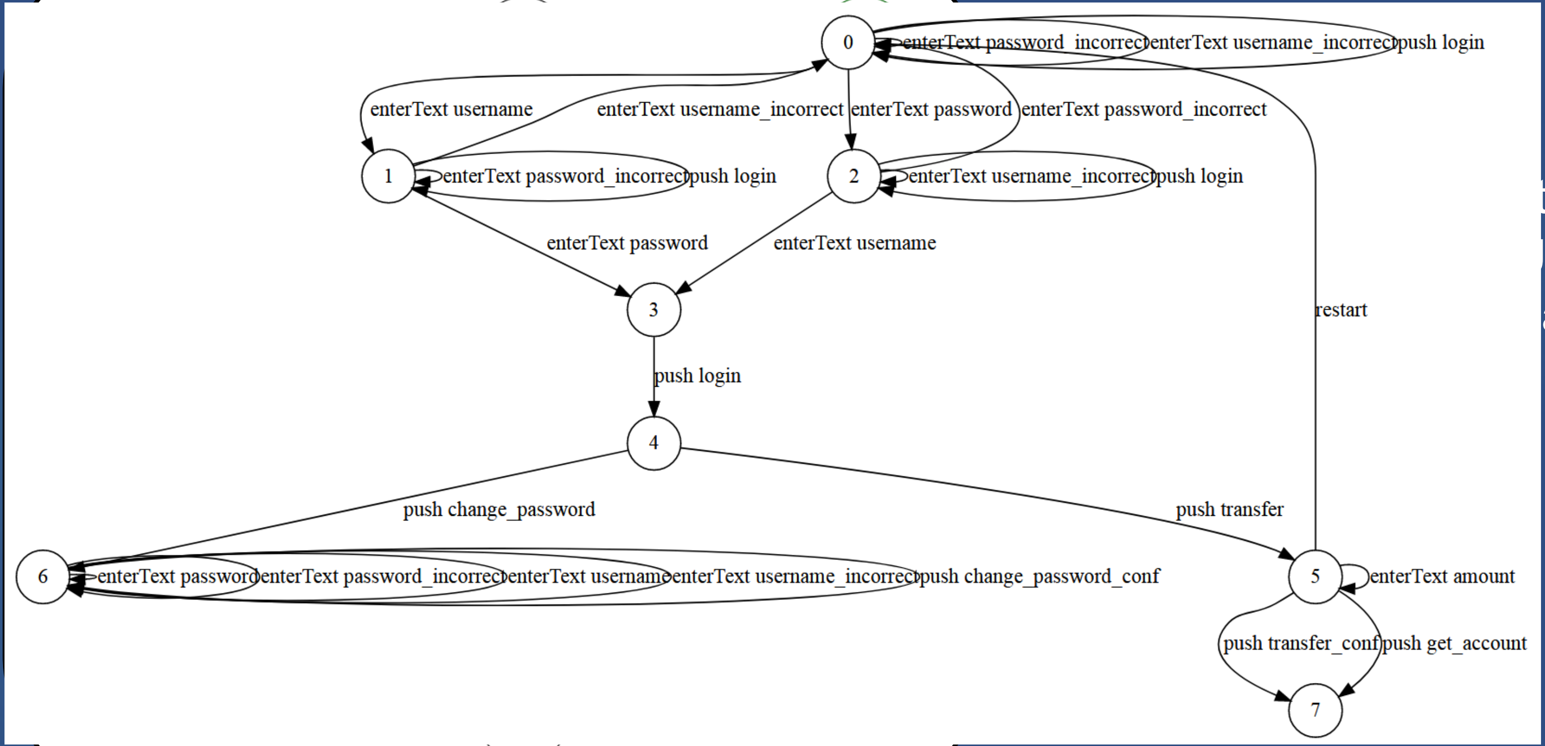
Results

- InsecureBankV2
  - Known Vulnerable App

- Fake WhatsApp
  - Adware
  - Malware Spreading



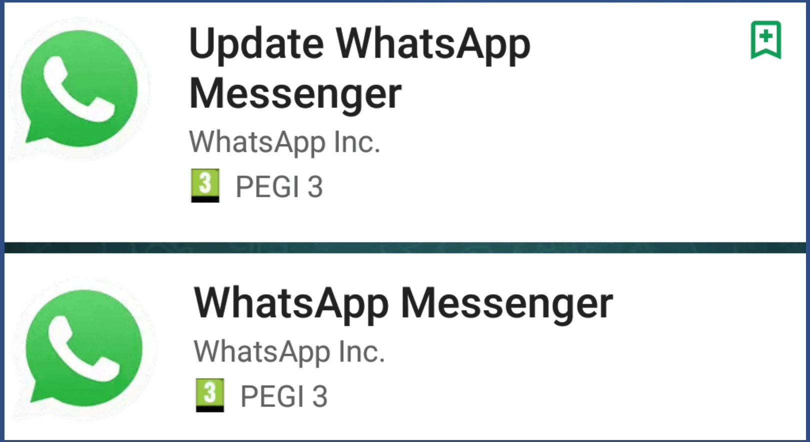
InsecureBankV2



tion
Jsage
ation

Fake WhatsApp

Background: Unicode trick allows malicious developers to impersonate WhatsApp developers
> 1M downloads, now deleted from Play Store



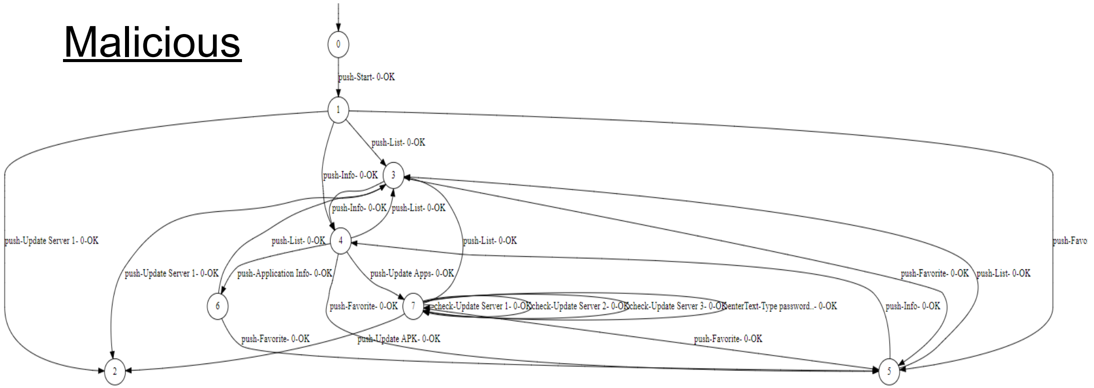
Fake WhatsApp

- Extraneous Functionality
  - SUT > REF: 100%
  - REF > SUT: 100%

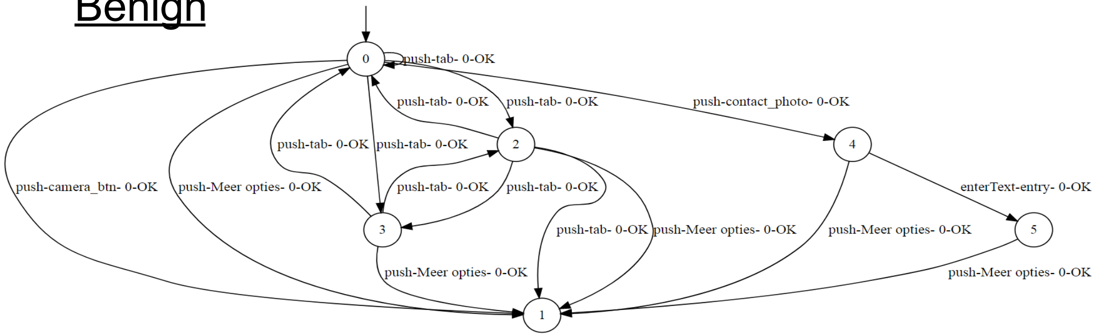
- Insecure Communication

```
GET http://req.startappserv
ice.com/1.4/..
```

Malicious



Benign



Run statistics

Learning Algorithm	L*
Equivalence Oracle	RandomWalk
Membership Queries	1778
Equivalence Queries	1
States	9
Transitions	25
Learning Time	26:06 $(hh:mm)$

Tab. 4.1: Statistics for active learning the inferred state machine for the 9292 application using L* and RandomWalk

Learning Algorithm	TTT
Equivalence Oracle	RandomWalk
Membership Queries	55
Equivalence Queries	2
States	2
Transitions	4
Learning Time	00:21 $(hh:mm)$

Tab. 4.2: Statistics for active learning the inferred machine for the 9292 application using TTT and RandomWalk

Learning Algorithm	TTT	L*
Equivalence Oracle	WMethod-Minimal	WMethod-Minimal
Membership Queries	15619	16966
Equivalence Queries	10	2
States	10	10
Transitions	30	30
Learning Time	23:14 $(hh:mm)$	27:37 $(hh:mm)$

Tab. 4.3: Statistics for Active Learning the Inferred Machine for the 9292 application using TTT, L* and WMethod-Minimal

Run statistics

InsecureBankv2	
Learning Time	3:13
Learning Algorithm	TTT
Equivalence Oracle	WMethod-Minimal
Equivalence Queries	7
Membership Queries	11627
States	6
Alphabet Size	6
Cache Hit	49%
Fast Forwarded	42%

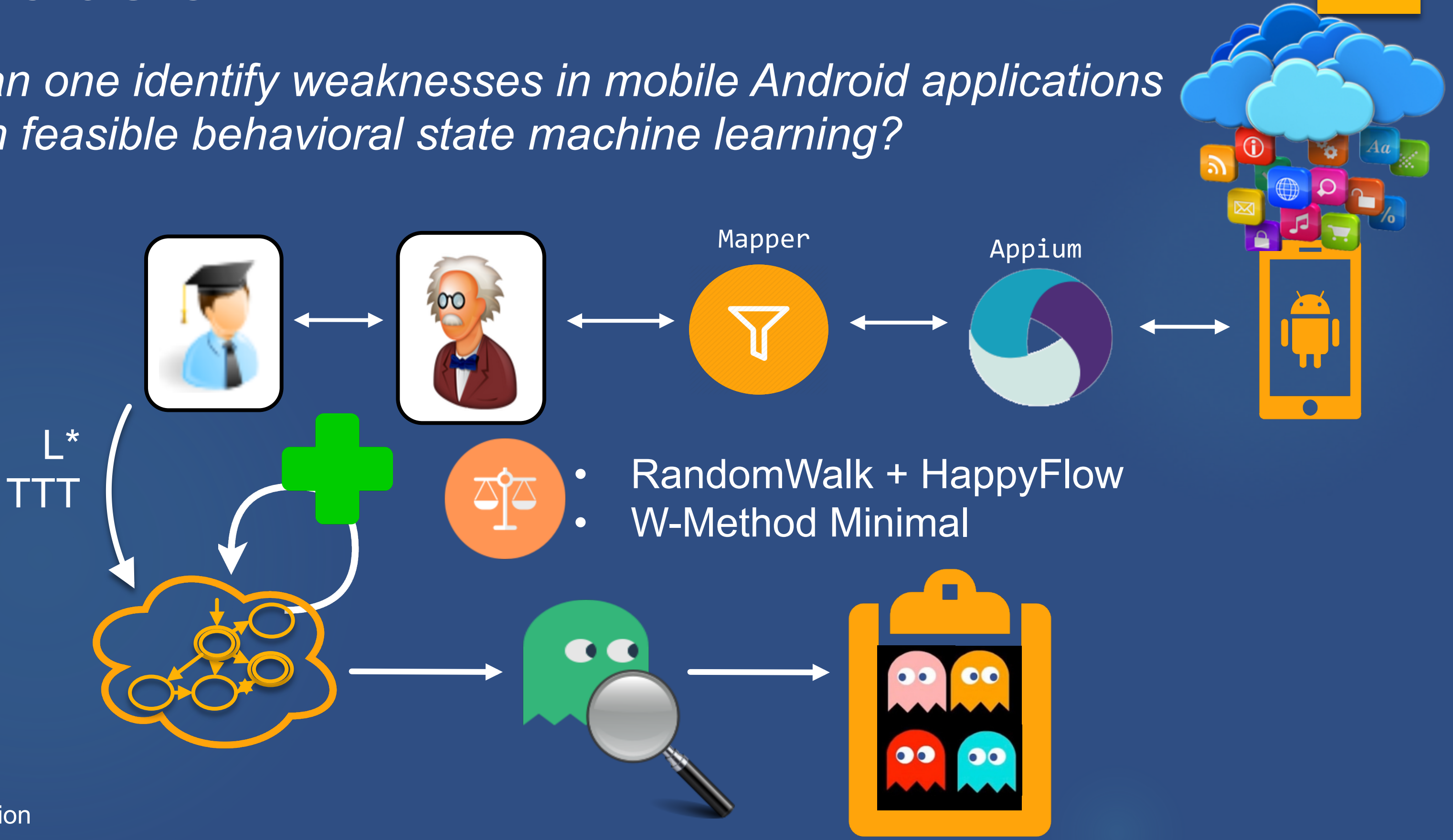
Tab. 6.1: Learning statistics for InsecureBankv2

	Fake WhatsApp	Real WhatsApp
Learning Time	7:22	5:47
Learning Algorithm	TTT	TTT
Equivalence Oracle	WMethod-Minimal	WMethod-Minimal
Equivalence Queries	7	6
Membership Queries	29.751	25.083
States	22	11
Alphabet Size	12	11
Cache Hit	71%	69%
Fast Forwarded	67%	62%

Tab. 6.3: Learning statistics for both WhatsApp versions

Conclusion

How can one identify weaknesses in mobile Android applications through feasible behavioral state machine learning?



Future work

- It is feasible, but slow, possible speedups:
  - Obtaining the possible actions without testing
  - Using other information, e.g. the screen content, during learning
  - …
- Performing the security analysis using a model checker
- Solving inconsistencies caused by popups or the keyboard not closing

Thank You

```
S = E = {λ}
(S,E,T) ← MQ(λ) ∪ ∀a ∈ A : MQ(a) # (S,E,T) is the observation table

While M is incorrect: # M is the conjecture
  While (S,E,T) is not consistent or not closed
    if (S,E,T) is not consistent:
        ∃(s1, s2) ∈ S, a ∈ A, e ∈ E : row(s1) = row(s2) and T(s1 · a · e) ≠ T(s2 · a · e)
        E ← a · e
        extend T to (S ∪ S · A) · E using MQ
    if (S,E,T) is not closed:
        ∃s1 ∈ S, a ∈ A : ∀ row(s1 · a) ≠ row(s)
        S ← s1 · a
        extend T to (S ∪ S · A) · E using MQ
  M = M(S,E,T)  # (S,E,T) is closed and consistent
  if Teacher replies with a counter−example t:
    add t and all its prefixes to S
    extend T to (S ∪ S · A) · E using MQ
Halt and output M
```

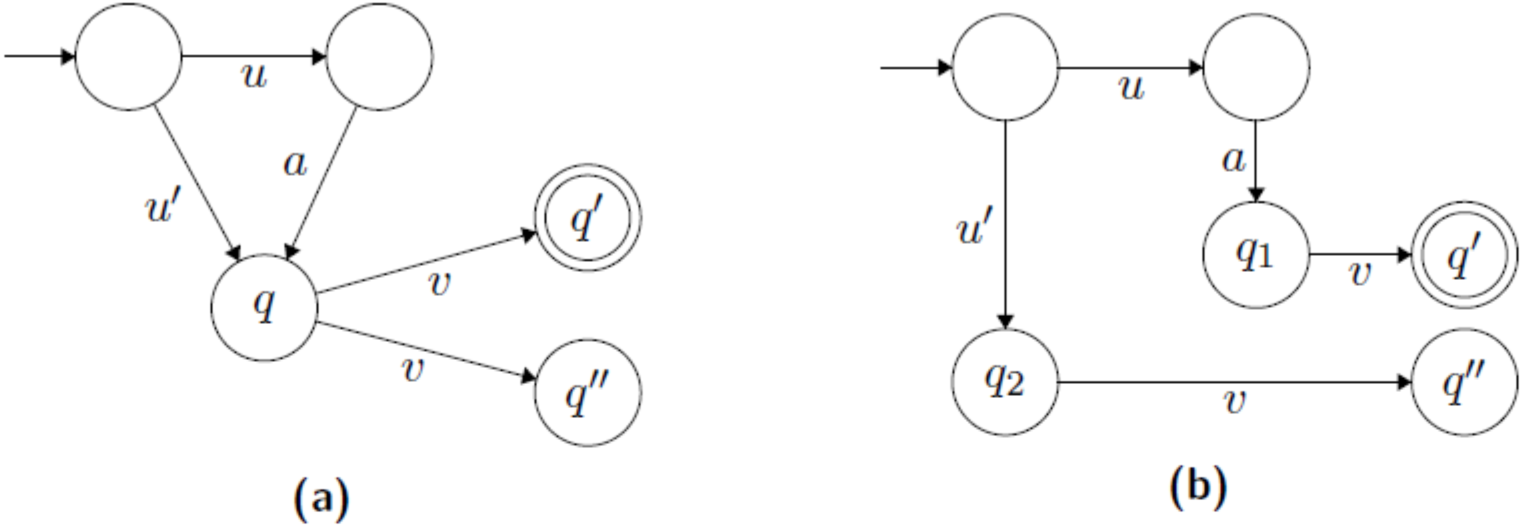


Fig. 2.4: Formal progression of an incorrect conjecture: (a) inconsistent model for distinguishing suffix v from state q, (b) consistent model after splitting q into new states q_1 and q_2.

	Improper Platform Usage	Insecure Communication	Insecure Authentication	Extraneous Functionality
InsecureBank	+/+	+/+	+/+	−/−
FreeCola	+/−	−/−	−/+	−/−
9292	−/−	−/−	−/−	−/−
Benign WhatsApp	−/−	−/−	−/−	−/−
Malicious WhatsApp	+/+	+/+	−/−	+/+

Tab. 7.2: The expected results versus the expected results for all applications under test.

Algorithm 7 Improper Platform Usage Identification

Input: Inferred State Machine $M = \{Q, \Sigma, \delta, q_0, F\}$

Output: Returns a set R of activities that induce supplementary behavior in machine M. There possibly exists a vulnerability if R is non-empty.

$R \leftarrow \emptyset$
$A \leftarrow$ activities from M
for all a in A **do**
    if a is a callable activity **then**
        add a to R
    end if
end for ▷ R contains all callable activities
for all q in Q and R is not empty **do**
    remove q's activity from R
end for
return R

Algorithm 8 Insecure Communication Identification

Input: Inferred State Machine $M = \{Q, \Sigma, \delta, q_0, F\}$

Output: Returns a set R of requests that made by actions in machine M that do not adhere to common network security standards. There possibly exists a vulnerability if R is non-empty.

$R \leftarrow \emptyset$
for all t in δ **do**
    $r \leftarrow$ request_insecure(t.request)
    if r **then**
        add r to R
    end if
end for
return R

Algorithm 9 Insecure Authentication Identification

Input: Inferred State Machine $M = \{Q, \Sigma, \delta, q_0, F\}$
Output: Returns a set R of authentication bypass techniques in machine M. There possibly exists a vulnerability if R is non-empty.

$a \leftarrow$ authentication state of M ▷ $a \in Q$
if a is `null` **then** ▷ no authentication → no authentication bypass
    return R
end if
$Marks \leftarrow$ subset of nodes possible to reach after a
$M' \leftarrow M - a$ ▷ the machine without the authentication state
for all m in $Marks$ **do**
    if a path from the q_0 to m exists in M' **then**
        add *path* to R
    end if
end for
$Q'' \leftarrow Q - Marks$
$A \leftarrow$ callable activities
for all q in Q'' and A is not empty **do**
    remove q's activity from A
end for
add A to R
return R

Algorithm 10 Code Tampering Identification

Input: Inferred machine $M = \{Q, \Sigma, \delta, q_0, F\}$ and reference machine $M' = \{Q', \Sigma', \delta', q_0', F'\}$
Output: Finds difference in M and M' Returns a set R of sequences that yield a different output for the two machines M and M'. The sequences are divided into sets R_1 and R_2 which depict what machine models the sequence. There possibly exists a vulnerability if R is non-empty.

$R_1, R_2 \leftarrow \emptyset$
$TCS_1 \leftarrow TCS(M)$
$TCS_2 \leftarrow TCS(M')$
for all $w \in TCS_1$ **do**
    if $\lambda^M(w) \neq \lambda^{M'}(w)$ **then**
        $R_1 \leftarrow w$
    end if
end for
for all $w \in TCS_2$ **do**
    if $\lambda^M(w) \neq \lambda^{M'}(w)$ and $w \notin R$ **then**
        $R_2 \leftarrow w$
    end if
end for
$R \leftarrow R_1, R_2$
return R