OblivIO:
Securing reactive programs by oblivious execution with bounded traffic overheads

Jeppe Fredsgaard Blaabjerg
Aslan Askarov
Aarhus University
Traffic analysis

Example

```plaintext
channel ALICE/ERROR: (int*int);
var balance: int[];

TRANSFER(from: int, amount: int, to: int) {
    if amount <= balance[from] then {
        balance[from] -= amount;
        balance[to] += amount;
    } else send(ALICE/ERROR,(amount, balance[from]));
}
```
Traffic analysis

Other observable properties of online communication

- Message timing

- Message size

- Message recipient
Mitigating traffic analysis

System-level mitigation

- Black-box
- Constant rate traffic of fixed-size packets
- Prohibitive overheads in practice: traffic or latency

---

Example
What is the right system-level bandwidth?

channel FORWARD: int;
var cnd: int;

RELAY(x: int) {
  if cnd
    then send(FORWARD,x);
  else skip;
}

- Traffic padding only needed if cnd is secret
  - Not known at system level
Mitigating traffic analysis

Which messages are sensitive?

\[ \approx_L \]

Which messages are sensitive?

Depend on secrets

Independent of secrets
Mitigating traffic analysis

Idea: Traffic padding guided by program source

Dummy
Real

\[\approx_L\]
OblivIO

Language and syntax

- Simple imperative language for reactive programs
- Data-oblivious execution model — Control-flow is never secret
  - Execution mode real or phantom can be secret
- Formal model includes computational history for computing timestamp

\[
p ::= \cdot \mid ch(x)\{c\}; p
\]
\[
c ::= \text{skip} \mid c_1; c_2 \mid x = e \mid \text{if } e \text{ then } c \text{ else } c \mid \text{while } e \text{ do } c \mid \text{send}(ch, e)
\]
\[
\mid x ?= e \quad (* \text{Oblivious, padding assignment} *)
\]
\[
\mid \text{oblif } e \text{ then } c \text{ else } c \quad (* \text{Oblivious conditional — executes both branches} *)
\]
\[
\mid x ?= \text{input}(ch,e) \quad (* \text{Local input} *)
\]


Oblivious semantics

Control flow

Oblivious conditional

$b$ is a stack of execution mode bits $b$
- $b = 1$ denotes real mode
- $b = 0$ denotes phantom mode

Real

![Real Oblivious Conditional Diagram]

Phantom

![Phantom Oblivious Conditional Diagram]
Oblivious semantics

Assignment

Oblivious assignment

Real

1 :: \(\overline{b}, x \mapsto (\text{"Goodbye"})_7\)

\(x \not= \text{"Hello"};\)

1 :: \(\overline{b}, x \mapsto (\text{"Hello"})_7\)

Phantom

0 :: \(\overline{b}, x \mapsto (\text{"Hello"})_5\)

\(x \not= \text{"Goodbye"};\)

0 :: \(\overline{b}, x \mapsto (\text{"Hello"})_7\)
Oblivious semantics

Sending

Real

1 :: $\overline{b}, x \mapsto (v)_z$

\[\text{send(ch, x);} \leadsto ch_1((v)_z)\]

1 :: $\overline{b}, x \mapsto (v)_z$


Phantom

0 :: $\overline{b}, x \mapsto (v)_z$

\[\text{send(ch, x);} \leadsto ch_0((v)_z)\]

0 :: $\overline{b}, x \mapsto (v)_z$

Dummy
Type system

Part a

\[ T-\text{If} \]
\[
\Gamma; \Delta \vdash e : \text{int@}_e \quad \Gamma, \Pi, \Lambda; \Delta; \text{pc} \vdash c_1 \\
\Gamma, \Pi, \Lambda; \Delta; \text{pc} \vdash \text{if } e \text{ then } c_1 \text{ else } c_2
\]

\[ \text{Public guard} \]

\[ T-\text{OblivIf} \]
\[
\ell \neq \bot \\
\Gamma, \Pi, \Lambda; \Delta; \text{pc} \cup \ell \vdash c_1 \\
\Gamma, \Pi, \Lambda; \Delta; \text{pc} \cup \ell \vdash c_2
\]

\[ \text{Non-public guard} \]

\[ T-\text{Assign} \]
\[
x \notin \text{dom}(\Delta) \\
\Gamma(x) = \sigma@\ell_x \\
\Gamma, \Pi, \Lambda; \Delta; \bot \vdash^q x = e
\]

\[ \text{Public pc} \]

\[ T-\text{OblivAssign} \]
\[
x \notin \text{dom}(\Delta) \\
\Gamma(x) : \sigma@\ell_x \\
\Gamma, \Pi, \Lambda; \Delta; \text{pc} \vdash x = e
\]

\[ \text{Any pc} \]

\[ T-\text{Send} \]
\[
\Gamma; \Delta \vdash e : \sigma@\ell_e \\
\Lambda(ch) = \sigma@\ell_\text{mode}; \ell_\text{val}
\]

\[
\text{pc} \subseteq \ell_\text{mode} \\
\ell_e \subseteq \ell_\text{val}
\]

\[ \Gamma, \Pi, \Lambda; \Delta; \text{pc} \vdash \text{send}(ch, e) \]
Theorem

Soundness

- Soundness theorem
  - Well-typed OblivIO programs do not leak by their traffic patterns

\[ k(cfg, \tau, \ell) \triangleq \{ cfg_2 \mid cfg \approx_{\ell} cfg_2 \land cfg_2 \xrightarrow{\star} \tau_2 \land \approx_{\ell} \tau_2 \} \]

Attacker knowledge\(^4\)

\[ k(cfg, \tau \cdot \alpha, \ell) \supseteq k(cfg, \tau, \ell) \]

Security condition

---

Example

Example 1 revisited

channel ERROR\textsubscript{H} : (\text{int} \times \text{int})\textsubscript{H};
var balance: \text{int}[]\textsubscript{H};

TRANSFER\textsubscript{L}(from: \text{int}\textsubscript{H}, amount: \text{int}\textsubscript{H}, to: \text{int}\textsubscript{H}) {
  obliv amount \leq balance[from]
  then {
    balance[from] -= amount;
    balance[to] += amount;
  }
  else send(ERROR, (amount, balance[from]));
}
Problem

Unbounded number of dummy messages

channel pong_H: int_H;

PING_H (x: int_H) {
    oblf x
    then send(PONG,1);
    else send(PONG,0);
}

channel ping_H: int_H;

PONG_H (x: int_H) {
    oblf x
    then send(PING,1);
    else send(PING,0);
}
Problem

Unbounded number of dummy messages

channel pong\textsubscript{H}: int\textsubscript{H};

channel ping\textsubscript{H}: int\textsubscript{H};

PING\textsubscript{H} (x: int\textsubscript{H}) {  
oblif x  
then send(PONG,1);  
else send(PONG,0);  
}

PONG\textsubscript{H} (x: int\textsubscript{H}) {  
oblif x  
then send(PING,1);  
else send(PING,0);  
}
Problem

Unbounded number of dummy messages

```
channel pong_H: int_H;

PING_H (x: int_H) {
    oblif x
    then send(PONG,1);
    else send(PONG,0);
}

channel ping_H: int_H;

PONG_H (x: int_H) {
    oblif x
    then send(PING,1);
    else send(PING,0);
}
```

Message queue
Problem

Unbounded number of dummy messages

channel pong_h: int_h;

PONG_h (x: int_h) {
    oblif x
    then send (PONG, 1);
    else send (PONG, 0);
}

channel ping_h: int_h;
PONG_h (x: int_h) {
    oblif x
    then send (PING, 1);
    else send (PING, 0);
}

Message queue

Message queue
Solution

Resource tracking in type-system

- Declare integer potential $q$ of a handler
  - Spend potential when sending obliviously
    - Oblivious send on channel with potential $r$ costs $1 + r$
      - 1 to pay for the message itself
      - $r$ to pay for the potential of the handler
- Instrument typing judgements with potentials
Type system

Adding potentials

\[
\text{T-If} \quad \Gamma; \Delta \vdash e : \text{int@}\ell \quad \Gamma, \Pi, \Lambda; \Delta; pc \vdash c_1 \quad \Gamma, \Pi, \Lambda; \Delta; pc \vdash c_2
\]

\[
\frac{}{\Gamma, \Pi, \Lambda; \Delta; pc \vdash \text{if } e \text{ then } c_1 \text{ else } c_2}
\]

\[
\text{T-OblivIf} \quad \Gamma; \Delta \vdash e : \text{int@}\ell
\]

\[
\frac{\ell \neq \bot}{\Gamma, \Pi, \Lambda; \Delta; pc \cup \ell \vdash c_1}
\]

\[
\frac{}{\Gamma, \Pi, \Lambda; \Delta; pc \vdash \text{oblif } e \text{ then } c_1 \text{ else } c_2}
\]

\[
\text{T-Send} \quad \Gamma; \Delta \vdash e : \sigma@\ell_e
\]

\[
\frac{}{\Lambda(ch) = \sigma@\ell_{\text{mode}}; \ell_{\text{val}}}
\]

\[
\text{pc} \subseteq \ell_{\text{mode}} \quad \ell_e \subseteq \ell_{\text{val}}
\]

\[
\frac{}{\Gamma, \Pi, \Lambda; \Delta; pc \vdash \text{send}(ch, e)}
\]
Type system
Adding potentials

T-If
$$\Gamma ; \Delta \vdash e : \text{int@}\ell \quad \Gamma , \Pi , \Lambda ; \Delta ; pc \vdash^q c_1 \quad \Gamma , \Pi , \Lambda ; \Delta ; pc \vdash^q c_2$$
$$\Gamma , \Pi , \Lambda ; \Delta ; pc \vdash^q \text{if } e \text{ then } c_1 \text{ else } c_2$$

T-OblivIf
$$\Gamma ; \Delta \vdash e : \text{int@}\ell$$
$$\ell \neq \bot \quad \Gamma , \Pi , \Lambda ; \Delta ; pc \cup \ell \vdash^{q_1} c_1 \quad \Gamma , \Pi , \Lambda ; \Delta ; pc \cup \ell \vdash^{q_2} c_2$$
$$\Gamma , \Pi , \Lambda ; \Delta ; pc \vdash^{q_1 + q_2} \text{oblif } e \text{ then } c_1 \text{ else } c_2$$

T-Send
$$\Gamma ; \Delta \vdash e : \sigma \circ \ell'_e$$
$$\Lambda(ch) = \sigma \circ \ell'_{mode} \circ \ell'_{val} ; r$$
$$pc \subseteq \ell'_{mode} \quad \ell'_e \subseteq \ell'_{val} \quad q' = \begin{cases} 0 & \text{if } pc = \bot \\ 1 + r & \text{otherwise} \end{cases}$$
$$\Gamma , \Pi , \Lambda ; \Delta ; pc \vdash^{q+q'} \text{send}(ch, e)$$
Theorem

Overhead

- Given
  - (System-wide) OblivIO trace $\tau_1$
  - (System-wide) Unpadded trace $\tau_2$
    - Without *dummy* messages
- Then
  - $|\tau_1| \leq |\tau_2| \times c$
Example

Example 2 revisited

channel pong_{H} \texttt{$M$: int}_{H};

\texttt{PING}_{H} \texttt{$N$ (x: int}_{H}) \{ 
    \texttt{oblif} \ x 
    \texttt{then} \ \texttt{send}(\texttt{PONG},1); 
    \texttt{else} \ \texttt{send}(\texttt{PONG},0); 
\}

\texttt{$N=2+2*\texttt{$M$}}

channel ping_{H} \texttt{$N$: int}_{H};

\texttt{PONG}_{H} \texttt{$M$ (x: int}_{H}) \{ 
    \texttt{oblif} \ x 
    \texttt{then} \ \texttt{send}(\texttt{PING},1); 
    \texttt{else} \ \texttt{send}(\texttt{PING},0); 
\}

\texttt{$M=2+2*\texttt{$N$}}
Discussion

Limitations

- Events are network messages only
  - Cannot react to events with secret presence
- Constant-time implementation of all operations
- Programs are static
  - No dynamically registered handlers
  - Functions not first-class
- Channels not first-class

```c
oblif secret
then ch ?= ALICE/GREET;
else ch ?= BOB/GREET;
send(ch,"Hello");
```
Summary

Mitigating traffic analysis with OblivIO

- Message presence
  - Sending dummy messages under *phantom* mode
- Message timing
  - Data-obliveness ensuring constant-time execution
- Message size
  - Padding value size at oblivious assignments
- Message recipient
  - Channels given in program text
Conclusion

Takeaways

- OblivIO
  - Secures reactive programs by oblivious execution
    - Well-typed programs do not leak by traffic patterns (Theorem 1)
  - Bound on the traffic overhead
    - Every real message generates at most $\epsilon$ dummy messages (Theorem 2)

Thank you!
Related work

Traffic analysis


Related work

Constant-time execution and data-obliviousness

Related work

Resource analysis


