Mining Task Precedence Graphs from Real-Time Embedded System Traces

Oleg Iegorov  Sebastian Fischmeister

Department of Electrical and Computer Engineering
University of Waterloo, Canada
{oiegorov, sfischme}@uwaterloo.ca

April 13, 2018
Software Growth in Real-time Systems

Sources:

Software-based Recalls in Automotive

Busy Beaver

$S(n, m)$: the largest number of steps taken by an $n$-state, $m$-symbol machine started on an initially blank tape before halting.
Busy Beaver

\( S(n, m) \): the largest number of steps taken by an \( n \)-state, \( m \)-symbol machine started on an initially blank tape before halting.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1RB</td>
<td>1LA</td>
</tr>
<tr>
<td>1</td>
<td>1LB</td>
<td>1RH</td>
</tr>
</tbody>
</table>

http://bit.ly/IJF9d0
**Busy Beaver**

\( S(n, m) \): the largest number of steps taken by an \( n \)-state, \( m \)-symbol machine started on an initially blank tape before halting.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1RB</td>
<td>1LA</td>
</tr>
<tr>
<td>1</td>
<td>1LB</td>
<td>1RH</td>
</tr>
</tbody>
</table>


0 0 1 1 1 1 0 0 (6 steps, four "1"s total)
Busy Beaver

$$S(5, 2) =$$

![Busy Beaver diagram]

- **State Diagram**
  - **A**
  - **B**
  - **C**
  - **D**
  - **E**
  - **H**

- **Transitions**
  - From **A**:
    - 0:1L -> E
    - 1:0L -> E
  - From **B**:
    - 0:1R -> C
    - 1:1L -> C
    - 1:1R -> A
  - From **C**:
    - 0:1R -> D
    - 1:1L -> D
  - From **D**:
    - 0:1L -> E
    - 1:1R -> E
  - From **E**:
    - 0:1L -> A
    - 1:0L -> A
    - 1:0R -> C
  - **H** is the halting state.
Busy Beaver

\[ S(5, 2) = \]

\[ S(2, 5) = \]

\[ \geq 47, 176, 870 \]
Busy Beaver

\[ S(5, 2) = \]

\[ S(2, 5) = \geq 1.9 \times 10^{704} \]

\[ \geq 47, 176, 870 \]
Need better theory, methods, and tools to build *and understand* systems.
Consequence from Current Trends

Need better theory, methods, and tools to build and understand systems.

→ Reverse engineering
Consequence from Current Trends

Need better theory, methods, and tools to build *and understand* systems.

→ Reverse engineering

**Good news:** more complexity means more data!
Consequence from Current Trends

Need better theory, methods, and tools to build \textit{and understand} systems.

→ Reverse engineering

\textbf{Good news:} more complexity means more data!

→ \textbf{Data-driven} reverse engineering
“(Software) reverse engineering is the process of analyzing a subject system to create representations of the system at a higher level of abstraction” [Chikofsky et al, 1990]

Traditionally used in the domain of desktop and enterprise software
Uses of Reverse Engineering in Real-time Systems

Possible applications:

- legacy code maintenance
- debugging
- anomaly detection
- testing
- documentation
Possible applications:

- legacy code maintenance
- debugging
- anomaly detection
- testing
- documentation

Timing is neglected in standard software reverse engineering tools.
Uses of Reverse Engineering in Real-time Systems

Possible applications:

• legacy code maintenance
• debugging
• anomaly detection
• testing
• documentation

Timing is neglected in standard software reverse engineering tools.

→ Tools for reverse engineering for real-time systems are needed
Characteristics of a Reverse Engineering Tool

- Representation of a system
  - Finite state machine
  - Petri net
  - Regular expressions
  - UML model
  - and others...

- Extraction of information from a system
  - Static: from source code
  - Dynamic: from traces
  - Hybrid

- Learning behaviour
  - Active: create new test cases and observe reaction
  - Passive: only observe
Characteristics of a Reverse Engineering Tool

- **Representation of a system**
  - Finite state machine
  - Petri net
  - Regular expressions
  - UML model
  - *Task precedence graph* ← TPG_miner
  - and others...

- **Extraction of information from a system**
  - Static: from source code
  - Dynamic: from traces
  - Hybrid

- **Learning behaviour**
  - Active: create new test cases and observe reaction
  - Passive: only observe
Characteristics of a Reverse Engineering Tool

• Representation of a system
  • Finite state machine
  • Petri net
  • Regular expressions
  • UML model
  • Task precedence graph $\leftarrow$ TPG_miner
  • and others...

• Extraction of information from a system
  • Static: from source code
  • Dynamic: from traces $\leftarrow$ TPG_miner
  • Hybrid

• Learning behaviour
  • Active: create new test cases and observe reaction
  • Passive: only observe
Characteristics of a Reverse Engineering Tool

- Representation of a system
  - Finite state machine
  - Petri net
  - Regular expressions
  - UML model
  - Task precedence graph $\leftarrow$ TPG_miner
  - and others...

- Extraction of information from a system
  - Static: from source code
  - Dynamic: from traces $\leftarrow$ TPG_miner
  - Hybrid

- Learning behaviour
  - Active: create new test cases and observe reaction
  - Passive: only observe $\leftarrow$ TPG_miner
TPG miner (Task Precedence Graph miner)

Real-Time Embedded System

Trace

Task Precedence Graph

τ1
τ2 τ3 τ4 τ5 τ6
τ7 τ8
τ9
τ10 τ11 τ12 τ13

Sebastian Fischmeister, sfischme@uwaterloo.ca
- **Task ID**: conjunction of fields that uniquely identify a task.
- **Event**: tuple `<timestamp, task ID>`
- **Trace**: chronologically ordered list of events
Task Precedence Graph (TPG)

TPG is a DAG:

- Nodes ($\tau_i$): task IDs;
- Edges ($\tau_i \rightarrow \tau_j$): immediate precedence relations ($\tau_i$’s output is $\tau_j$’s input), $\tau_i \ll \tau_j$. 

Problem: How to mine immediate precedence relations from a trace?

Sebastian Fischmeister, sfischme@uwaterloo.ca
TPG is a DAG:
- Nodes ($\tau_i$): task IDs;
- Edges ($\tau_i \rightarrow \tau_j$): immediate precedence relations ($\tau_i$'s output is $\tau_j$'s input), $\tau_i \ll \tau_j$.

Problem: How to mine immediate precedence relations from a trace?
1. Transactionalize trace to different events
1. Transactionalize trace to different events
2. Identify occurrence of events between events (→ occurrence pattern)
TPG_Miner Approach

1. Transactionalize trace to different events
2. Identify occurrence of events between events (→ occurrence pattern)
3. Extract precedence relations
TPG_Miner Approach

1. Transactionalize trace to different events
2. Identify occurrence of events between events (→ occurrence pattern)
3. Extract precedence relations
4. Create DAG from relations
TPG_Miner Approach

1. Transactionalize trace to different events
2. Identify occurrence of events between events (→ occurrence pattern)
3. Extract precedence relations
4. Create DAG from relations
5. Clean up the DAG
Occurrence Pattern (op)

Given a string $s$, its **occurrence pattern**, $op(s)$, is the shortest substring of $s$ that "covers" $s$:

$$s = \langle 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0 \rangle$$
Occurrence Pattern (op)

Given a string $s$, its **occurrence pattern**, $op(s)$, is the shortest substring of $s$ that "covers" $s$:

$$s = \langle 1,0,0,1,0,1,0,0,1,0 \rangle$$

- $p_1 = \langle 1,0,0,1,0 \rangle$ is the occurrence pattern of $s$, $op(s) = p_1$
Given a string $s$, its **occurrence pattern**, $op(s)$, is the shortest substring of $s$ that "covers" $s$:

$$s = \langle 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0 \rangle$$

- $p_1 = \langle 1, 0, 0, 1, 0 \rangle$ is the occurrence pattern of $s$, $op(s) = p_1$
- $p_2 = \langle 3, 2 \rangle$ is not a substring of $s$
Occurrence Pattern (op)

Given a string \( s \), its **occurrence pattern**, \( op(s) \), is the shortest substring of \( s \) that "covers" \( s \):

\[
s = \langle 1,0,0,1,0,1,0,0,1,0,1,0,0,1,0,1,0,0 \rangle
\]

- \( p_1 = \langle 1,0,0,1,0 \rangle \) is the occurrence pattern of \( s \), \( op(s) = p_1 \)
- \( p_2 = \langle 3,2 \rangle \) is not a substring of \( s \)
- \( p_3 = \langle 1,0 \rangle \) is a substring of \( s \), doesn’t cover \( s \)
Given a string $s$, its **occurrence pattern**, $op(s)$, is the shortest substring of $s$ that "covers" $s$:

$$s = \langle 1,0,0,1,0,1,0,0,1,0,1,0,0,1,0 \rangle$$

- $p_1 = \langle 1,0,0,1,0 \rangle$ is the occurrence pattern of $s$, $op(s) = p_1$
- $p_2 = \langle 3,2 \rangle$ is not a substring of $s$
- $p_3 = \langle 1,0 \rangle$ is a substring of $s$, doesn’t cover $s$
- $p_4 = \langle 1,0,0,1,0,1,0,0,1,0,1,0 \rangle$ covers $s$, but not the shortest (see $p_1$).
Occurrence Pattern (op)

Given a string \( s \), its occurrence pattern, \( op(s) \), is the shortest substring of \( s \) that "covers" \( s \):

\[
 s = \langle 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0 \rangle
\]

- \( p_1 = \langle 1, 0, 0, 1, 0 \rangle \) is the occurrence pattern of \( s \), \( op(s) = p_1 \)
- \( p_2 = \langle 3, 2 \rangle \) is not a substring of \( s \)
- \( p_3 = \langle 1, 0 \rangle \) is a substring of \( s \), doesn't cover \( s \)
- \( p_4 = \langle 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0 \rangle \) covers \( s \), but not the shortest (see \( p_1 \)).

Given a trace \( T = \langle \tau_i, \tau_h, \ldots \rangle \), \( op(\tau_i, \tau_j) \) is the occurrence pattern of a string \( s \) whose elements are the numbers of occurrences of \( \tau_i \) between consecutive occurrences of \( \tau_j \).

\[
 T = \langle \tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_7, \tau_3, \tau_4, \tau_7, \tau_5, \tau_4, \tau_7, \tau_2, \tau_1, \tau_5, \tau_7, \tau_6, \tau_7, \tau_7, \tau_3, \tau_2, \tau_5, \tau_4, \tau_7, \tau_7, \tau_5, \tau_2, \tau_7, \tau_1, \tau_7, \tau_2, \tau_7, \tau_5, \tau_1, \tau_7, \tau_3 \rangle
\]

\[
op(\tau_2, \tau_7) = \langle 1, 0, 0, 1, 0 \rangle
\]
Theorem

If a task $\tau_i$ is an immediate predecessor of a task $\tau_j$ ($\tau_i \ll \tau_j$) in a real-time system $S$, then there exists $op(\tau_i, \tau_j)$ in any execution trace of $S$ captured during at least two hyperperiods of $S$.

Visualization of occurrences of tasks $\tau_2$ and $\tau_7$ during two hyperperiods;

$\tau_2 \ll \tau_7$; $op(\tau_2, \tau_7) = \langle 1, 0, 0, 1, 0 \rangle$. 
Function mine_ops($T$)

Data: trace $T$
Result: $\text{op}(\tau_i, \tau_j) \forall \tau_i, \tau_j \in T$

foreach $\tau_i \in T$ do
  foreach $\tau_j \in T$, $i \neq j$ do
    $s \leftarrow$ numbers of occurrences of $\tau_j$ between consecutive occurrences of $\tau_i$ in $T$
    foreach prefix $q$ of $s$, $\text{length}(q)=1, 2, \ldots, \text{length}(s)/2$ do
      if $q$ covers $s$ then
        $\text{op}(\tau_i, \tau_j) \leftarrow q$
        break
      else if $\text{length}(q)==\text{length}(s)/2$ then
        $\text{op}(\tau_i, \tau_j) \leftarrow \text{NULL}$
      end
    end
  end
return $\text{op}$
end

$s = \langle 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0 \rangle$
Function \texttt{mine_ops}(T)

Data: trace \( T \)

Result: \( \text{op}(\tau_i, \tau_j) \ \forall \tau_i, \tau_j \text{ in } T \)

\begin{align*}
&\text{foreach } \tau_i \in T \text{ do} \\
&\quad \text{foreach } \tau_j \in T, i \neq j \text{ do} \\
&\quad \quad s \leftarrow \text{numbers of occurrences of } \tau_j \text{ between consecutive occurrences of } \tau_i \text{ in } T \\
&\quad \quad \text{foreach prefix } q \text{ of } s, \text{length}(q)=1,2,\ldots,\text{length}(s)/2 \text{ do} \\
&\quad \quad \quad \text{if } q \text{ covers } s \text{ then} \\
&\quad \quad \quad \quad \text{op}(\tau_i, \tau_j) \leftarrow q \\
&\quad \quad \quad \quad \text{break} \\
&\quad \quad \text{else if } \text{length}(q)==\text{length}(s)/2 \text{ then} \\
&\quad \quad \quad \quad \text{op}(\tau_i, \tau_j) \leftarrow \text{NULL} \\
&\quad \text{end} \\
&\text{end} \\
&\text{return } \text{op} \\
&\text{end}
\end{align*}

\[ s = \langle 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0 \rangle \]

\[ q = \langle 1 \rangle \]
Function \texttt{mine_ops}(T)

Data: trace $T$

Result: $op(\tau_i, \tau_j) \forall \tau_i, \tau_j$ in $T$

\begin{align*}
\text{foreach } \tau_i \in T \text{ do} & \\
& \hspace{1em} \text{foreach } \tau_j \in T, i \neq j \text{ do} & \\
& & s \leftarrow \text{numbers of occurrences of } \tau_j \text{ between consecutive occurrences of } \tau_i \text{ in } T & \\
& & \text{foreach prefix } q \text{ of } s, \text{length}(q)=1,2,...,\text{length}(s)/2 \text{ do} & \\
& & & \text{if } q \text{ covers } s \text{ then} & \\
& & & & \hspace{1em} op(\tau_i, \tau_j) \leftarrow q & \\
& & & & \text{break} & \\
& & & \text{else if } \text{length}(q)==\text{length}(s)/2 \text{ then} & \\
& & & & \hspace{1em} op(\tau_i, \tau_j) \leftarrow \text{NULL} & \\
& & \text{end} & \\
& \text{end} & \\
& \text{return } op & \\
\end{align*}

$s = \langle 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0 \rangle$

$q = \langle 1, 0 \rangle$
Mining Occurrence Patterns in a Trace

Function `mine_ops(T)`

Data: trace $T$

Result: $\text{op}(\tau_i, \tau_j) \ \forall \tau_i, \tau_j \in T$

foreach $\tau_i \in T$ do
  foreach $\tau_j \in T$, $i \neq j$ do
    $s \leftarrow$ numbers of occurrences of $\tau_j$ between consecutive occurrences of $\tau_i$ in $T$
    foreach prefix $q$ of $s$, $\text{length}(q)=1,2,...,\text{length}(s)/2$ do
      if $q$ covers $s$ then
        $\text{op}(\tau_i, \tau_j) \leftarrow q$
        break
      else if $\text{length}(q)==\text{length}(s)/2$ then
        $\text{op}(\tau_i, \tau_j) \leftarrow \text{NULL}$
      end
    end
  end
end

$s = \langle 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0 \rangle$
$q = \langle 1, 0, 0 \rangle$
Function \texttt{mine\_ops}(T)

Data: trace \( T \)

Result: \( op(\tau_i, \tau_j) \ \forall \tau_i, \tau_j \) in \( T \)

\begin{verbatim}
foreach \( \tau_i \in T \) do
    foreach \( \tau_j \in T, i \neq j \) do
        \( s \leftarrow \) numbers of occurrences of \( \tau_j \) between consecutive occurrences of \( \tau_i \) in \( T \)
        foreach prefix \( q \) of \( s \), length\((q) = 1, 2, \ldots, length(s) / 2 \) do
            if \( q \) covers \( s \) then
                \( op(\tau_i, \tau_j) \leftarrow q \)
                break
            else if \( length(q) == length(s) / 2 \) then
                \( op(\tau_i, \tau_j) \leftarrow \) NULL
            end
        end
    end
end
\end{verbatim}

\( s = \langle 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0 \rangle \)

\( q = \langle 1, 0, 0, 1 \rangle \)
Function `mine_ops(T)`

Data: trace $T$

Result: $op(\tau_i, \tau_j) \forall \tau_i, \tau_j \in T$

foreach $\tau_i \in T$ do
  foreach $\tau_j \in T$, $i \neq j$ do
    $s \leftarrow$ numbers of occurrences of $\tau_j$ between consecutive occurrences of $\tau_i$ in $T$
    foreach prefix $q$ of $s$, $\text{length}(q) = 1, 2, \ldots, \text{length}(s) / 2$ do
      if $q$ covers $s$ then
        $op(\tau_i, \tau_j) \leftarrow q$
        break
      else if $\text{length}(q) = \text{length}(s) / 2$ then
        $op(\tau_i, \tau_j) \leftarrow \text{NULL}$
    end
  end
end

return $op$

\[ s = \langle 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0 \rangle \]
\[ q = \langle 1, 0, 0, 1, 0 \rangle \]
Function $\text{mine}_\text{TPG}(T)$

Data: trace $T$

Result: TPG of $T$

1. $\text{nodes} \leftarrow \text{unique task IDs in } T$
2. $\text{edges} \leftarrow \text{mine}_\text{ops}(T)$
3. $\text{TPG} \leftarrow (\text{nodes}, \text{edges})$
4. $\text{TPG} \leftarrow \text{transitive}_\text{closure}(\text{TPG})$
5. return $\text{TPG}$
Function `mine_TPG(T)`

Data: trace $T$

Result: TPG of $T$

1. `nodes ← unique task IDs in $T$`
2. `edges ← mine_ops($T$)`
3. `TPG ← (nodes, edges)`
4. `TPG ← transitive_closure(TPG)`
5. `return TPG`

---

![Diagram of TPG](image-url)
Mining a TPG from a Trace

**Function** `mine_TPG(T)`

**Data:** trace $T$

**Result:** TPG of $T$

1. nodes $\leftarrow$ unique task IDs in $T$
2. edges $\leftarrow$ `mine_ops(T)`
3. $\text{TPG} \leftarrow$ (nodes, edges)
4. $\text{TPG} \leftarrow$ transitive_closure($\text{TPG}$)
5. return $\text{TPG}$
Function mine_TPG(T)
   Data: trace T
   Result: TPG of T
   nodes ← unique task IDs in T
   edges ← mine_ops(T)
   TPG ← (nodes, edges)
   TPG ← transitive_closure(TPG)
   return TPG
Anomaly-based Intrusion Detection Systems

- Intrusion detection system (IDS) monitors a system/network for malicious activity.
- Anomaly-based IDS identifies observations (packets, syscalls, etc.) which do not conform to an expected pattern observed during normal operation.
- IDS must be trustworthy, low false positive rate is often more important than low false negative rate.
- Desired features of an IDS:
  - online operation
  - explainable results
- TPG represents an expected pattern for an anomaly-based IDS.
Train TPG on other traces and perform online anomaly detection:

**Data:** trace stream $W$, TPG $G$

**Result:** trained $G$ or anomalous event $\epsilon$ in $W$ for each $\epsilon \in W$ in the increasing order of $\epsilon$. time

1. $\nu \leftarrow \epsilon$.
2. **task id**
3. foreach $\tau$ such that $e(\tau, \nu) \in G$
4. $u(\tau, \nu) \leftarrow$ append the # of occurrences of $\tau$ since previous occurrence of $\nu$
5. if $u(\tau, \nu)$ is not a prefix of $op(\tau, \nu)$
6. if training then remove $e(\tau, \nu)$ from $G$; update $G$
7. else return $\epsilon$. time, $u(\tau, \nu)$, $op(\tau, \nu)$
8. end
9. if training then return $G$
10. else return False

$T = \langle op(\tau_2, \tau_7) = \langle 1, 0, 0, 1, 0 \rangle, u(\tau_2, \tau_7) = \langle 0 \rangle \rangle$
Train TPG on other traces and perform online anomaly detection:

Data: trace stream $\mathcal{W}$, TPG $G$
Result: trained $G$ or anomalous event $\epsilon$ in $\mathcal{W}$

foreach $\epsilon \in \mathcal{W}$ in the increasing order of $\epsilon$.time do
  $\nu \leftarrow \epsilon$.task_id
  foreach $\tau$ such that $e(\tau, \nu) \in G$ do
    $u(\tau, \nu) \leftarrow$ append the # of occurrences of $\tau$ since previous occurrence of $\nu$
    if $u(\tau, \nu)$ is not a prefix of $op(\tau, \nu)$ then
      if training then remove $e(\tau, \nu)$ from $G$; update $G$
      else return $\epsilon$.time, $u(\tau, \nu)$, $op(\tau, \nu)$
    end
  end
  if training then return $G$
  else return False
end

$T = \langle$
$op(\tau_2, \tau_7) = \langle 1, 0, 0, 1, 0 \rangle$
$u(\tau_2, \tau_7) = \langle$
TPGs for Anomaly Detection

Train TPG on other traces and perform online anomaly detection:

Data: trace stream $W$, TPG $G$

Result: trained $G$ or anomalous event $\varepsilon$ in $W$

foreach $\varepsilon \in W$ in the increasing order of $\varepsilon$.time do
    $\upsilon \leftarrow \varepsilon$.task_id
    foreach $\tau$ such that $e(\tau, \upsilon) \in G$ do
        $u(\tau, \upsilon) \leftarrow$ append the # of occurrences of $\tau$ since previous occurrence of $\upsilon$
        if $u(\tau, \upsilon)$ is not a prefix of $op(\tau, \upsilon)$ then
            if $training$ then remove $e(\tau, \upsilon)$ from $G$; update $G$
            else return $\varepsilon$.time, $u(\tau, \upsilon)$, $op(\tau, \upsilon)$
        end
    end
    if $training$ then return $G$
else return False
end

$T = \langle \tau_1, \tau_2, \rangle$

$op(\tau_2, \tau_7) = \langle 1, 0, 0, 1, 0 \rangle$

$u(\tau_2, \tau_7) = \langle 1 \rangle$
TPGs for Anomaly Detection

Train TPG on other traces and perform online anomaly detection:

Data: trace stream $W$, TPG $G$
Result: trained $G$ or anomalous event $\varepsilon$ in $W$

\[
\text{foreach } \varepsilon \in W \text{ in the increasing order of } \varepsilon.\text{time} \text{ do}
\]
\[
\nu \leftarrow \varepsilon.\text{task_id}
\]
\[
\text{foreach } \tau \text{ such that } e(\tau, \nu) \in G \text{ do}
\]
\[
\nu(\tau, \nu) \leftarrow \text{append the } \# \text{ of occurrences of } \tau \text{ since previous occurrence of } \nu
\]
\[
\text{if } \nu(\tau, \nu) \text{ is not a prefix of } op(\tau, \nu) \text{ then}
\]
\[
\text{if training then remove } e(\tau, \nu) \text{ from } G; \text{ update } G
\]
\[
\text{else return } \varepsilon.\text{time}, \nu(\tau, \nu), op(\tau, \nu)
\]
\[
\text{end}
\]
\[
\text{if training then return } G
\]
\[
\text{else return False}
\]
\[
T = \langle \tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_7 \rangle
\]
\[
op(\tau_2, \tau_7) = \langle 1, 0, 0, 1, 0 \rangle
\]
\[
u(\tau_2, \tau_7) = \langle 1 \rangle
\]
\[
u(\tau_2, \tau_7) \text{ is a prefix of } op(\tau_2, \tau_7)
\]
TPGs for Anomaly Detection

Train TPG on other traces and perform online anomaly detection:

Data: trace stream $W$, TPG $G$
Result: trained $G$ or anomalous event $\varepsilon$ in $W$

\[
\begin{align*}
\text{foreach } \varepsilon \in W \text{ in the increasing order of } \varepsilon.\text{time do} \\
&\quad \upsilon \leftarrow \varepsilon.\text{task_id} \\
&\quad \text{foreach } \tau \text{ such that } e(\tau, \upsilon) \in G \text{ do} \\
&\quad & u(\tau, \upsilon) \leftarrow \text{append the } \# \text{ of occurrences of } \tau \text{ since previous occurrence of } \upsilon \\
&\quad & \text{if } u(\tau, \upsilon) \text{ is not a prefix of } op(\tau, \upsilon) \text{ then} \\
&\quad & \quad \text{if } \text{training then remove } e(\tau, \upsilon) \text{ from } G; \text{ update } G \\
&\quad & \quad \text{else return } \varepsilon.\text{time}, u(\tau, \upsilon), op(\tau, \upsilon) \\
&\quad \text{end} \\
&\quad \text{if } \text{training then return } G \\
&\text{else return } \text{False} \\
\end{align*}
\]

\[
T = \langle \tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_7, \tau_3, \tau_4, \tau_7 \rangle \\
op(\tau_2, \tau_7) = \langle 1, 0, 0, 1, 0 \rangle \\
u(\tau_2, \tau_7) = \langle 1, 0 \rangle \quad \text{\(u(\tau_2, \tau_7)\) is a prefix of \(op(\tau_2, \tau_7)\)}
TPGs for Anomaly Detection

Train TPG on other traces and perform online anomaly detection:

Data: trace stream $W$, TPG $G$
Result: trained $G$ or anomalous event $\varepsilon$ in $W$

foreach $\varepsilon \in W$ in the increasing order of $\varepsilon$.time do
  $\tau \leftarrow \varepsilon$.task_id
  foreach $\tau$ such that $e(\tau, \tau) \in G$ do
    $u(\tau, \tau) \leftarrow$ append the # of occurrences of $\tau$ since previous occurrence of $\tau$
    if $u(\tau, \tau)$ is not a prefix of $op(\tau, \tau)$ then
      if training then remove $e(\tau, \tau)$ from $G$; update $G$
      else return $\varepsilon$.time, $u(\tau, \tau)$, $op(\tau, \tau)$
    end
  end
  if training then return $G$
else return False

$T = \langle \tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_7, \tau_3, \tau_4, \tau_7, \tau_5, \tau_2 \rangle$

$op(\tau_2, \tau_7) = \langle 1, 0, 0, 1, 0 \rangle$

$u(\tau_2, \tau_7) = \langle 1, 0, 1 \rangle$
TPGs for Anomaly Detection

Train TPG on other traces and perform online anomaly detection:

Data: trace stream $\mathbb{W}$, TPG $G$

Result: trained $G$ or anomalous event $\varepsilon$ in $\mathbb{W}$

\[
\text{foreach } \varepsilon \in \mathbb{W} \text{ in the increasing order of } \varepsilon.\text{time do}
\]

\[
\text{foreach } \tau \text{ such that } e(\tau, \varepsilon) \in G \text{ do}
\]

\[
\upsilon \leftarrow \varepsilon.\text{task_id}
\]

\[
\text{foreach } \tau \text{ such that } e(\tau, \upsilon) \in G \text{ do}
\]

\[
\upsilon(\tau, \upsilon) \leftarrow \text{append the # of occurrences of } \tau \text{ since previous occurrence of } \upsilon
\]

\[
\text{if } \upsilon(\tau, \upsilon) \text{ is not a prefix of } \text{op}(\tau, \upsilon) \text{ then}
\]

\[
\text{if training then remove } e(\tau, \upsilon) \text{ from } G; \text{ update } G
\]

\[
\text{else return } \varepsilon.\text{time}, \upsilon(\tau, \upsilon), \text{op}(\tau, \upsilon)
\]

\[
\text{end if training then return } G
\]

\[
\text{else return False}
\]

\[
T = \langle \tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_7, \tau_3, \tau_4, \tau_7, \tau_5, \tau_2, \tau_4, \tau_2 \rangle
\]

\[
op(\tau_2, \tau_7) = \langle 1, 0, 0, 1, 0 \rangle
\]

\[
u(\tau_2, \tau_7) = \langle 1, 0, 2 \rangle
\]
TPGs for Anomaly Detection

Train TPG on other traces and perform online anomaly detection:

Data: trace stream $\mathcal{W}$, TPG $G$
Result: trained $G$ or anomalous event $\varepsilon$ in $\mathcal{W}$

foreach $\varepsilon \in \mathcal{W}$ in the increasing order of $\varepsilon$.time do
  $\tau \leftarrow \varepsilon$.task_id
  foreach $\tau$ such that $e(\tau, \varepsilon) \in G$ do
    $u(\tau, \varepsilon) \leftarrow$ append the # of occurrences of $\tau$ since previous occurrence of $\varepsilon$
    if $u(\tau, \varepsilon)$ is not a prefix of $op(\tau, \varepsilon)$ then
      if training then remove $e(\tau, \varepsilon)$ from $G$; update $G$
      else return $\varepsilon$.time, $u(\tau, \varepsilon)$, $op(\tau, \varepsilon)$
    end
  end
if training then return $G$
else return False
end

$T = \langle \tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_7, \tau_3, \tau_4, \tau_7, \tau_5, \tau_2, \tau_4, \tau_2, \tau_7 \rangle$

$op(\tau_2, \tau_7) = \langle 1, 0, 0, 1, 0 \rangle$

$u(\tau_2, \tau_7) = \langle 1, 0, 2 \rangle$

$u(\tau_2, \tau_7)$ is not a prefix of $op(\tau_2, \tau_7)$
Modern vehicles:

- becoming connected (internet, bluetooth, radars, etc.);
- becoming autonomous (automatic parking, autopilot).

Security threats:

1. Remotely hack one or several ECUs, run malicious code on them.
2. Spoof the identity of the hacked ECUs.
3. Take over the control of the vehicle.

Examples:

- Miller and Valasek on Jeep Cherokee (2015)\(^1\)
- Tencent Keen Lab on Tesla Model S (2016, 2017)\(^2\)

\(^1\)https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/
\(^2\)https://www.pcmag.com/news/355281/tesla-model-s-hackers-return-for-encore-attack
Modern vehicles:

- becoming connected (internet, bluetooth, radars, etc.);
- becoming autonomous (automatic parking, autopilot).

Security threats:

1. Remotely hack one or several ECUs, run malicious code on them.
2. Spoof the identity of the hacked ECUs.
3. Take over the control of the vehicle.

Examples:

- Miller and Valasek on Jeep Cherokee (2015) \(^1\)
- Tencent Keen Lab on Tesla Model S (2016, 2017) \(^2\)

\(^1\)https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/
\(^2\)https://www.pcmag.com/news/355281/tesla-model-s-hackers-return-for-encore-attack
Controller Area Network (CAN) bus:

- de-facto standard for interconnecting vehicle’s ECUs;
- ECUs broadcast messages;
- no authentication;

CAN bus trace:

<table>
<thead>
<tr>
<th>time</th>
<th>message ID</th>
<th>bytes</th>
<th>b1</th>
<th>b2</th>
<th>b3</th>
<th>b4</th>
<th>b5</th>
<th>b6</th>
<th>b7</th>
<th>b8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1478193190</td>
<td>0545</td>
<td>8</td>
<td>d8</td>
<td>5f</td>
<td>00</td>
<td>8b</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>1478193190</td>
<td>05f0</td>
<td>2</td>
<td>01</td>
<td>00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1478193190</td>
<td>0130</td>
<td>8</td>
<td>0b</td>
<td>80</td>
<td>00</td>
<td>ff</td>
<td>0f</td>
<td>80</td>
<td>0c</td>
<td>ea</td>
</tr>
<tr>
<td>1478193190</td>
<td>0131</td>
<td>8</td>
<td>f2</td>
<td>7f</td>
<td>00</td>
<td>00</td>
<td>15</td>
<td>7f</td>
<td>0c</td>
<td>35</td>
</tr>
<tr>
<td>1478193190</td>
<td>0140</td>
<td>8</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>1e</td>
<td>0d</td>
<td>2c</td>
<td>3b</td>
<td></td>
</tr>
<tr>
<td>1478193190</td>
<td>02c0</td>
<td>8</td>
<td>15</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>
Case Studies

CAN bus traces:

**Study 1**  Hyundai YF Sonata with injected and annotated spoofed messages (HCRL lab at Korea University)³

![Hyundai YF Sonata](image)

**Study 2**  Production vehicle exercised on a test track (traces provided by our partner).

³[http://ocslab.hksecurity.net/Datasets/CAN-intrusion-dataset](http://ocslab.hksecurity.net/Datasets/CAN-intrusion-dataset)
Case Study 1

- 4 traces with annotated spoofed messages.
- Spoofed messages have unique message IDs in two traces (DoS.csv and Fuzzy.csv):
  - Use these traces to train a TPG;
  - Train on "normal" parts: before the first spoofed message, 1000 events in each part.

![The trained TPG](image)

Sebastian Fischmeister, sfischme@uwaterloo.ca
Anomaly detection in the remaining 2 traces (gear.csv and RPM.csv):

- attacks were detected prior to the first spoofed message – very good:
  - 29 events before the first spoofed message in gear.csv
  - 7 events before the first spoofed message in RPM.csv
- detected anomalies are probably caused by the activation of an external board used to inject spoofed messages.
Case Study I (cont.)

Anomaly detection in the remaining 2 traces (gear.csv and RPM.csv):

• attacks were detected prior to the first spoofed message – very good:
  • 29 events before the first spoofed message in gear.csv
  • 7 events before the first spoofed message in RPM.csv
  • detected anomalies are probably caused by the activation of an external board used to
  inject spoofed messages.

• false positive rates:
  • gear.csv: 0.019 (19 out of 1000 events)
  • RPM.csv: 0.007 (7 out of 1000 events)
  • (almost) all false positives are returned in the beginning of traces.

Observation: synchronization between analyzer and runtime needed
Case Study II

- Traces capture normal behavior of vehicle’s ECUs, no anomalies
- The goal: evaluation of false positive rate
- 15 traces in total:

<table>
<thead>
<tr>
<th>Trace</th>
<th>Duration (s)</th>
<th># events</th>
<th># tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>312</td>
<td>793,653</td>
<td>252</td>
</tr>
<tr>
<td>2</td>
<td>245.5</td>
<td>653,932</td>
<td>240</td>
</tr>
<tr>
<td>3</td>
<td>361.3</td>
<td>907,038</td>
<td>224</td>
</tr>
<tr>
<td>4</td>
<td>390</td>
<td>1,044,558</td>
<td>251</td>
</tr>
<tr>
<td>5</td>
<td>352.6</td>
<td>891,195</td>
<td>235</td>
</tr>
<tr>
<td>6</td>
<td>287</td>
<td>727,508</td>
<td>243</td>
</tr>
<tr>
<td>7</td>
<td>288.7</td>
<td>767,148</td>
<td>239</td>
</tr>
<tr>
<td>8</td>
<td>307.2</td>
<td>1,260,299</td>
<td>243</td>
</tr>
<tr>
<td>9</td>
<td>600</td>
<td>1,425,667</td>
<td>219</td>
</tr>
<tr>
<td>10</td>
<td>343.2</td>
<td>866,187</td>
<td>229</td>
</tr>
<tr>
<td>11</td>
<td>319</td>
<td>847,882</td>
<td>241</td>
</tr>
<tr>
<td>12</td>
<td>249.9</td>
<td>635,463</td>
<td>235</td>
</tr>
<tr>
<td>13</td>
<td>342.1</td>
<td>991,105</td>
<td>276</td>
</tr>
<tr>
<td>14</td>
<td>314.3</td>
<td>834,942</td>
<td>257</td>
</tr>
<tr>
<td>15</td>
<td>283.8</td>
<td>754,786</td>
<td>239</td>
</tr>
</tbody>
</table>
Evaluate the number of false positives (FPs):

1) build a TPG $G$ using trace #1;
2) detect anomalies in trace #2 using $G$; detected anomalies are FPs.
3) refine $G$ with trace #2;
4) detect anomalies in trace #3 using $G$; detected anomalies are FPs.
5) refine $G$ with trace #3;
6) apply the same procedure on traces #4-15.

<table>
<thead>
<tr>
<th>Trace</th>
<th># FPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>140</td>
</tr>
<tr>
<td>3</td>
<td>220</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>38</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
</tr>
</tbody>
</table>

The trained TPG. Only nodes with I/O edges shown.
Many Other Applications

- Insight and understanding for developers
Many Other Applications

- Insight and understanding for developers
- Documentation of (legacy) systems
Many Other Applications

- Insight and understanding for developers
- Documentation of (legacy) systems
- Debugging
Many Other Applications

- Insight and understanding for developers
- Documentation of (legacy) systems
- Debugging
- Safety assessment
Many Other Applications

- Insight and understanding for developers
- Documentation of (legacy) systems
- Debugging
- Safety assessment
- Runtime monitoring
Many Other Applications

- Insight and understanding for developers
- Documentation of (legacy) systems
- Debugging
- Safety assessment
- Runtime monitoring
- Model generation for simulation and optimization
Many Other Applications

- Insight and understanding for developers
- Documentation of (legacy) systems
- Debugging
- Safety assessment
- Runtime monitoring
- Model generation for simulation and optimization
- ...

Sebastian Fischmeister, sfischme@uwaterloo.ca
Conclusion

• **TPG_miner**: an approach to reverse-engineer real-time embedded systems via mining task precedence graphs from system traces.
Conclusion

- **TPG**\text{\_miner}: an approach to reverse-engineer real-time embedded systems via mining task precedence graphs from system traces

- Evaluation in the context of automotive security
Conclusion

- **TPG_miner**: an approach to reverse-engineer real-time embedded systems via mining task precedence graphs from system traces

- Evaluation in the context of automotive security
  - can be used as an anomaly-based intrusion detection system

Sebastian Fischmeister, sfischme@uwaterloo.ca
**Conclusion**

- **TPG_miner**: an approach to reverse-engineer real-time embedded systems via mining task precedence graphs from system traces

- Evaluation in the context of automotive security
  - can be used as an anomaly-based intrusion detection system
  - straightforward application on CAN bus traces
Conclusion

- **TPG_miner**: an approach to reverse-engineer real-time embedded systems via mining task precedence graphs from system traces

- Evaluation in the context of automotive security
  - can be used as an anomaly-based intrusion detection system
  - straightforward application on CAN bus traces

- Detects anomalies in streaming data

Sebastian Fischmeister, sfischme@uwaterloo.ca 28 / 28
Conclusion

- **TPG_miner**: an approach to reverse-engineer real-time embedded systems via mining task precedence graphs from system traces

- Evaluation in the context of automotive security
  - can be used as an anomaly-based intrusion detection system
  - straightforward application on CAN bus traces

- Detects anomalies in streaming data

- Explains which expected system behavior has been violated
**Conclusion**

- **TPG** miner: an approach to reverse-engineer real-time embedded systems via mining task precedence graphs from system traces

- Evaluation in the context of automotive security
  - can be used as an anomaly-based intrusion detection system
  - straightforward application on CAN bus traces

- Detects anomalies in streaming data

- Explains which expected system behavior has been violated

- Requires little training to achieve low false positive rate
Conclusion

- **TPG_miner**: an approach to reverse-engineer real-time embedded systems via mining task precedence graphs from system traces

- Evaluation in the context of automotive security
  - can be used as an anomaly-based intrusion detection system
  - straightforward application on CAN bus traces

- Detects anomalies in streaming data

- Explains which expected system behavior has been violated

- Requires little training to achieve low false positive rate

- Implemented in R: [https://bitbucket.org/oiegorov/tpg_miner](https://bitbucket.org/oiegorov/tpg_miner)
Conclusion

- **TPG_miner**: an approach to reverse-engineer real-time embedded systems via mining task precedence graphs from system traces

- Evaluation in the context of automotive security
  - can be used as an anomaly-based intrusion detection system
  - straightforward application on CAN bus traces

- Detects anomalies in streaming data

- Explains which expected system behavior has been violated

- Requires little training to achieve low false positive rate

- Implemented in R: [https://bitbucket.org/oiegorov/tpg_miner](https://bitbucket.org/oiegorov/tpg_miner)
  - the README file explains how to reproduce results of Case Study I.