Cyber-Physical Systems Security: A Pacemaker Case Study

Partha Roop

BioEmulation Research Group
The University of Auckland

26 March 2021
Introduction
   Solution: Run-Time Verification

CPS Security using Run-Time Enforcement
   CPS attacks
   Run-time Enforcement

Correct Pacemaker Operation

Synchronous Discrete Timed Automata

Enforcers

Hardware Compilation

Results
This presentation is a result of collaborations with several colleagues and PhD students.

- Cooperation with Stavros Tripakis’s group at UC Berkeley and Aalto University on run-time verification and enforcement.

- The follow up cooperation with Srinivas Pinisetty and Gerardo Schneider from Chalmers in 2017 on Pacemaker security.

- Joint work with PhD student Hammond Pearce on CPS security for smart grids. We are exploring hardware security [1].

- The following publications are relevant for this presentation. [2]–[4].

---


Implantable Medical Devices (IMDs)

- Include pacemakers, defibrillators, insulin pumps, neurological pulse generators, ...

- **Safety-critical operation**: medical emergencies on malfunctions

---

Heart and Pacemaker communicate through 4 signals:
- **AS** and **VS** from the heart
- **AP** and **VP** from the pacemaker

Pacemaker ensures timing properties between signals.
IMDs are becoming “smarter” and more connected

- Increasingly complex sensors + software
- Wireless, internet-enabled features [6]

---

Unintended consequences

A new potential for malicious attacks

- “We are aware of hundreds of medical devices that have been infected by malware” — Bill Maisel, FDA [7]

- Notable examples:
  - Pacemakers which give deadly shocks to their patients [8]
  - Pumps remotely programmed to deliver incorrect insulin levels [9]
  - DoS attacks on implantable cardiac defibrillators [10]

---

Traditional security mechanisms suggested [11] but may not be suitable

- Low-power long-life devices may not be capable of de/encryption [12]
- SW updates not often provided due to regulatory framework [13]
- **In practice: impractical/infeasible to secure all attack vectors** [14]

---


Proposed solutions: Literature

Two general approaches

▶ Access Control (still has potential to be bypassed)
  ▶ E.g. Heart2Heart [15]
  ▶ E.g. Ultrasonic bounding [16]


Pacemaker Timing Requirements (EGMs)

InvasIC
Equivalent ECG timings

**P₁**  
*P* − *wave* and *R* − *wave* cannot happen simultaneously.

**P₂**  
*R* − *wave* must arrive within *PR* interval after a *P* − *wave*.

**P₃**  
*P* − *wave* must be true within *R* − *P* interval after an *R* − *wave*.

**P₄**  
After an *R* − *wave*, another *R* − *wave* can come only after *R* − *P* interval.

**P₅**  
After an *R* − *wave*, another *R* − *wave* should come within *R* − *R* interval.
Based on this information the condition of the leads can be found out, which will help in examining any malfunctions in the leads.

The pre-processing to remove unwanted data is done after the filtering of the ECG signal. The entire recording is divided into ECG cycles and then applied pre-processing to remove unwanted data.

After the filtering of the ECG signal, if the sampling frequency of the signal is corrupted with high frequency noise, Savitzky-Golay filtering is used to remove noise from the signal. MATLAB’s “filter design and analysis” application can also be used for filtering.

A huge collection of recorded physiologic signals is available from [19]. Consequently, the database of ECG signals for our experiments is also obtained from [19]. The ECG signal of the patient, who suffers from complete heart block and PVCs (Premature ventricular contractions), is considered. Additionally, our experiments also considered a patient with a pacemaker installed in his body. The ECG signals in use is from a 63 year old male patient.

The ECG signals in this paper include pacing artifacts and do not actually generate ECG signals. We use pacing artifacts and do not actually generate ECG signals. We use pacemakers to divide the entire recording into ECG cycles and then apply pre-processing to remove unwanted data. The entire recording is divided into ECG cycles and then applied pre-processing to remove unwanted data.

The pacemaker is wireless and can be used as a wearable device as this is secure (through secure authentication mechanisms such as human biometrics). However, monitoring these properties, implicitly prevent attacks.

Remark 4: If a pacemaker maliciously changes the programmed timing values, which may cause serious harm. However, the attacker is unable to access the wearable device and have not actually designed it in this paper. Thus, this paper demonstrates the technical feasibility of the wearable device and have not actually designed it in this paper.

In this section, we present the architecture of the proposed monitoring system which is hosted on the wearable device.

The ECG signal from the entire recorded signal is known. A comparison of the raw ECG signal and pre-processed signal is shown in Figure 9.

RV_Monitor Module takes the property to be verified \( \varphi \) and a stream of timed events (fed as input by the ECG processing module). These events are passed in the appropriate format (as timed events) and a stream of timed events (fed as input by the ECG processing module).

The RV_Monitor module runs in a loop, processing all ECG cycles in the recording one by one.

The RV_Monitor module is responsible for filtering the ECG signal. If the property to be verified \( \varphi \) is a timed automaton, the RV_Monitor module extracts all the relevant actions of interest for monitoring the ECG signal.

The RV_Monitor module outputs a verdict providing information whether the input event stream satisfies (resp. violates) property \( \varphi \).

Output: stream of verdicts.

Does \( \sigma \) satisfy \( \varphi \)?

\( \varphi \) is a timed automaton.

Output: stream of verdicts.
Definition (Timed automata)

A timed automaton $\mathcal{A} = (L, l_0, X, \Sigma, \Delta, F)$ is a tuple, s.t. $L$ is a finite set of locations with the initial location $l_0 \in L$, a finite set of clocks $X$, $\Sigma$ is a finite set of actions, $\Delta \subseteq L \times G(X) \times \Sigma \times 2^X \times L$ is the transition relation. $F \subseteq L$ is a set of accepting locations.

$$
\begin{align*}
\Sigma \setminus \{r\} & \quad \Sigma \setminus \{r\} & \quad \Sigma \\
\rightarrow l_0 & \quad r, \quad x := 0 & \quad r, \quad x < RP \\
\downarrow & \quad r, \quad x \geq RP, \quad x := 0 & \quad \uparrow
\end{align*}
$$
Overview of the solution

- Pacemaker timing parameters are programmed simultaneously on the monitoring device and the pacemaker.
- The wearable device monitors the familiar ECG to ensure that there have been no hacks.
- In the event of any timing violation an alarm is sounded.
Example P4

\[
\begin{align*}
\Sigma \setminus \{r\} & \quad \Sigma \setminus \{r\} & \quad \Sigma \\
\xrightarrow{r, x := 0} l_0 & \quad \xrightarrow{r, x < RP} l_1 & \quad \xrightarrow{r, x \geq RP, x := 0} l_2
\end{align*}
\]

Table: Property \( P_4 \) monitoring with \( RP = 900 \)

<table>
<thead>
<tr>
<th>( \sigma )</th>
<th>( M_\varphi(\sigma) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>((50, p))</td>
<td>( c_{true} )</td>
</tr>
<tr>
<td>((50, p) \cdot (208, r))</td>
<td>( c_{true} )</td>
</tr>
<tr>
<td>((50, p) \cdot (208, r) \cdot (300, p))</td>
<td>( c_{true} )</td>
</tr>
<tr>
<td>((50, p) \cdot (208, r) \cdot (300, p) \cdot (451, r))</td>
<td>( false )</td>
</tr>
</tbody>
</table>
Limitation of RV Monitors

- Monitoring can detect but is unable to intervene.
- CPS attacks are complex and vulnerabilities may be exploited more easily than conventional cyber security.
- Run-Time Enforcement has some interesting potential.
## Example Targeted Attacks

- **(2000)** Maroochy Shire wastewater attack, where raw sewage was released around a town by ex-employee.
- **(2006)** Los Angeles traffic system hack, disrupting four of the busiest intersections for days.
- **(2008)** Turkish pipeline explosion by suspected Russian operators to cut off oil to Georgia.
- **(2008)** Pacific Energy Resources SCADA attack, where system functions were impaired by ex-employee.
- **(2008)** Lodz, Poland, tram system was taken over by a teen hacker, causing injuries.
Example Targeted Attacks

- (2011) Springfield IL water distribution malfunction, pump destroyed, attributed to Romanian hacker.
- (2014) Unnamed German Steel mill, hackers caused massive damage to equipment by disabling shut-off procedures, including a blast furnace.
Example Targeted Attacks

- (2016) Unnamed water facility, where Syrian hacktivists took control of PLCs controlling toxic chemicals.
- (2016) San Francisco municipal rail system ransomware hack, free rides for commuters.
- (2017) Austrian ski resort ransomware hack, "smart locks" compromised, guests couldn’t access their rooms.
- (2017) Well-known WannaCry ransomware attack, which also infected hospital equipment such as MRI scanners, radiotherapy machines, oncology equipment etc.
- (2017) U.S. DHS reports govt. team hacking passenger jet controls.
Classifying attacks [17], [18]

In order to mitigate attacks, we must understand them.
Classifying attacks [17], [18]

In order to mitigate attacks, we must understand them.

### Passive Attacks

Exfiltrate data, gain knowledge of system, non-damaging.
In order to mitigate attacks, we must understand them.

**Passive Attacks**
Exfiltrate data, gain knowledge of system, non-damaging.

**Disruptive Attacks**

- Physical-Cyber Attack - originates in physical domain, aims to disrupt cyber domain, e.g. cutting cables.

- Cyber-Physical Attack - originates in cyberspace and impacts ability for cyber system to control physical process, e.g. DoS, Cryptolocker.

- Cyber-Kinetic Attack - originates in cyberspace and intends to cause tangible physical damage, e.g. Stuxnet.
## Classifying attacks [17], [18]

<table>
<thead>
<tr>
<th>Cyber-Physical</th>
<th>Cyber-Kinetic</th>
</tr>
</thead>
<tbody>
<tr>
<td>2006 LA Traffic</td>
<td>2000 Maroochy Shire Wastewater</td>
</tr>
<tr>
<td>2008 Pacific Energy Resources</td>
<td>2008 Turkish Pipeline</td>
</tr>
<tr>
<td>2016 Syrian Water Facility</td>
<td>2008 Lodz Trams</td>
</tr>
<tr>
<td>2016 San Francisco Rail</td>
<td>2009 Stuxnet</td>
</tr>
<tr>
<td>2017 Austrian Ski Resort</td>
<td>2011 Springfield Water Distribution</td>
</tr>
<tr>
<td>2017 WannaCry</td>
<td>2014 German Steel Mill</td>
</tr>
<tr>
<td></td>
<td>2015 Jeep Cherokee</td>
</tr>
<tr>
<td></td>
<td>2016 Tesla S</td>
</tr>
<tr>
<td></td>
<td>2017 Passenger Jet</td>
</tr>
</tbody>
</table>
Pacemaker Timing

1. AS
2. AR
3. AS
4. AS

Atrium

1. VS
2. VP
3. VP
4. VP

Ventricle

1. PVARP
2. VRP
3. VRP
4. PVARP

AVI

PVARP

VRP

AEI

LRI

URI

Extension

VP
Discretised Properties – $\varphi$

- Real valued times $AVI, AEI, URI, LRI$
- Discrete valued times $AVI_{cycles}, AEI_{cycles}, URI_{cycles}, LRI_{cycles}$

Properties

- $P_1$: $AP$ and $VP$ cannot happen simultaneously.
- $P_2$: $VS$ or $VP$ must be true within $AVI_{cycles}$ after an atrial event $AS$ or $AP$.
- $P_3$: $AS$ or $AP$ must be true within $AEI_{cycles}$ after a ventricular event $VS$ or $VP$.
- $P_4$: After a ventricular event, another ventricular event can happen only after $URI_{cycles}$.
- $P_5$: After a ventricular event, another ventricular event should happen within $LRI_{cycles}$. 
Proposed Approach

Heart (Plant)

Enforcer

Pacemaker (Controller)

Transformed Inputs

\[ \varphi \]

Transformed Outputs

Inputs

Transformed Outputs

Outputs

\[ AS \rightarrow VS \rightarrow AP' \rightarrow VP' \rightarrow Enforcer \rightarrow AS' \rightarrow VS' \rightarrow AP \rightarrow VP \]
Discrete Timed Automata — $\mathcal{A}$

A policy specification language from [19]

- Automata extended with integer variables as discrete clocks
- Discrete time more efficient than Dense time
- Clocks count in synchronous “ticks”

Example Property as DTA — $\mathcal{A}_\varphi$

$S_1$: “A and B cannot happen simultaneously, A and B alternate starting with an A. B should be true with in 5 ticks after A occurs.”

Deterministic DTA

- For any location $l$ and any two distinct transitions $(l, g_1, a, Y_1, l'_1) \in \Delta$ and $(l, g_2, a, Y_2, l'_2) \in \Delta$ with same source $l$, the conjunction of guards $g_1 \land g_2$ is unsatisfiable.

Complete DTA

- For any location $l \in L$ and any event $a \in \Sigma$, the disjunction of the guards of the transitions leaving $l$ and labelled by $a$ evaluates to $true$. 
Bidirectional enforcement requires a property solely reliant on inputs

Achieved by projecting DTA $\mathcal{A}_\varphi$ on inputs

- All locations and clocks remain
- All transitions remain, with outputs removed from guards
Enforcer Operation

- Enforcers operate **iteratively**
- They first edit inputs (if necessary) & emit, then edit outputs (if necessary) & emit.
- Then, they advance their internal DTA state.
Policy $P_4$: After a ventricular event $(VP \mid VS)$, a $VP$ may happen only after $URI_{cycles}$.

$\sum \setminus (VS \mid VP) \quad \sum \setminus (VP) \quad (VS \mid VP) \quad V := 0 \quad V \geq URI_{cycles}$

Assume $URI_{cycles} = 3$

<table>
<thead>
<tr>
<th>t</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VS'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>VP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>VP'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Edit Functions

Input Edit $- \text{edit}_{\varphi}^{I}(\sigma_I)$

- A set of possible next input events of an Input Word $(\sigma_I)$
- Such that the word can still be extended to satisfy the property $\varphi_I$

Output Edit $- \text{edit}_{\varphi}^{O}(\sigma, x)$

- A set of possible next output events of an Input-Output Word $(\sigma, x)$
- Such that the word can still be extended to satisfy the property $\varphi$

Variants

- Random Edit $- \text{rand-edit}_{\varphi}^{I}(\sigma_I)$ and $\text{rand-edit}_{\varphi}^{O}(\sigma, x)$
  - Randomly selects an element from the respective edit function

- Minimum Distance Edit $- \text{minD-edit}_{\varphi}^{I}(\sigma_I, x)$ and $\text{minD-edit}_{\varphi}^{O}(\sigma, x, y)$
  - Selects an element from the respective edit function with minimum distance from the current value
Why hardware?

The power of an enforcer

- Runtime enforcers are **omnipotent** — they can edit any I/O
- Potentially catastrophic if an enforcer is faulty or could be compromised

The consistency (security) of hardware

- Software can be altered/updated
- Hardware can be built to be permanent using ASICs/discrete components
- Software is intrinsically difficult to analyse
  - May require analysis of entire application/runtime/RTOS (*program could halt!*)
  - Requires processor model
- Hardware can more easily be checked for timing/functional properties
  - Requires analysis of just enforcer hardware module
Statically compiling synthesis algorithm

1: $t \leftarrow 0$
2: $q \leftarrow q_0$
3: while true do
4:   $x_t \leftarrow \text{read\_in\_chan}()$
5:   if $\exists \sigma'_i \in \Sigma^* : q \xrightarrow{x_t \cdot \sigma'_i} q' \land q' \in Q_F$ then
6:     $x'_t \leftarrow x_t$
7:   else
8:     $x'_t \leftarrow \text{rand\_edit}_{A_i}(q)$
9:   end if
10: release($x'_t$)
11: $y_t \leftarrow \text{read\_out\_chan}()$
12: if $\exists \sigma' \in \Sigma^* : q \xrightarrow{(x'_t, y_t) \cdot \sigma'} q' \land q' \in Q_F$ then
13:   $y'_t \leftarrow y_t$
14: else
15:   $y'_t \leftarrow \text{rand\_edit}_{O_i}(q, x'_t)$
16: end if
17: release($y'_t$)
18: $q \leftarrow q''$ where $q \xrightarrow{(x'_t, y'_t)} q''$ and $q'' \notin q_i$
19: $t \leftarrow t + 1$
20: end while
Constraint: Next State $q' \notin q_v$
Verifying the hardware

Functional correctness using EBMC [20]

- Security vulnerabilities can be present in implementations of otherwise-correct systems (e.g. Heartbleed [21])
- EBMC is a model checker for hardware designs.
  - It functions over assertions in Verilog Code.
- We can assert $\forall q \in Q, \forall (x, y) \in \Sigma, E(q, x, y) \rightarrow (q', x', y')$ such that $q \notin q_v$.
  - i.e. EBMC will check the combinatorial update implementation for the possibility of any input at any time that could cause a transition to a violation state.
- We use k-induction with k=1.
- As it is analysing a combinatorial function, the analysis is very quick.

[21]. S. Inc. (2017), Heartbleed bug,
Verifying the hardware

Timing correctness using Quartus TimeQuest [22]

▶ Quartus TimeQuest will determine the critical path and max $f_{clk}$ of the system.
▶ As there are two registers for signals to propagate through, the overhead is $\frac{1}{f_{clk}} \times 2$

Power consumption using Quartus PowerPlay [23]

▶ Assume $f_{clk} = 100\text{kHz} = 10\mu\text{S}$, so overhead = $20\mu\text{S}$.
▶ I/O toggle rate set at average of 1.5 transitions/S (avg. 90 bpm).
▶ Vectorless estimation for internal signals (more pessimistic).

[22]. TimeQuest timing analyzer: Quick start tutorial, UG-TMQSTANZR-1.1, Altera, Dec. 2009
Additional hardware risk assessment

▶ Failure rate of system is not $\text{fail(} \text{enforcer} \text{)} \times \text{fail(} \text{pacemaker} \text{)}$.

▶ Enforcer encapsulates original controller and will take over in failure scenario.

▶ Failure rate of system is just $\text{fail(} \text{enforcer} \text{)}$.

Attacker modelling

▶ Policies $P_1$ through $P_5$ effectively mitigate attack scenarios
  ▶ Attacker switches off pacing? ($P_2, P_3, P_5$)
  ▶ Attacker reprograms pacemaker to pace too fast? ($P_4$)
  ▶ Attacker reprograms pacemaker to pace $\text{AP}$ and $\text{VP}$ simultaneously? ($P_1$)

▶ EBMC validates that all attack traces are mitigated for safe minimum QoS.
Hardware Synthesis Results

Experimental Methodology

- Policies provided for $P_1$ through $P_5$
- Enforcer Verilog synthesized with Intel Quartus 16.0 to Max V CPLD
- EBMC verifies enforcer constraint
- Quartus TimeQuest provides information
- Quartus PowerPlay can estimate CPLD power consumption
## Results: HW consumption

<table>
<thead>
<tr>
<th>Enforcer Policy</th>
<th>States</th>
<th>Timers</th>
<th>Transitions</th>
<th>LEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>158</td>
</tr>
<tr>
<td>$P_3$</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>158</td>
</tr>
<tr>
<td>$P_4$</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>158</td>
</tr>
<tr>
<td>$P_5$</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>158</td>
</tr>
<tr>
<td>$P_{1,2,3,4}$</td>
<td>5</td>
<td>2</td>
<td>13</td>
<td>335</td>
</tr>
<tr>
<td>$P_{1,2,3,4,5}$</td>
<td>5</td>
<td>2</td>
<td>19</td>
<td>343</td>
</tr>
<tr>
<td>$P_1 \land P_2 \land P_3 \land P_4$</td>
<td>9</td>
<td>3</td>
<td>84</td>
<td>494</td>
</tr>
<tr>
<td>$P_1 \land P_2 \land P_3 \land P_4 \land P_5$</td>
<td>17</td>
<td>4</td>
<td>304</td>
<td>761</td>
</tr>
</tbody>
</table>

### Analysis

- Increasing complexity (States, Timers, Transitions) $\rightarrow$ more hardware (LEs)
## Results: HW performance

<table>
<thead>
<tr>
<th>Enforcer Policy</th>
<th>LEs</th>
<th>Verification Time (s)</th>
<th>Min OH (ns)</th>
<th>Dynamic Power (mW @ 100kHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>8</td>
<td>&lt;0.01</td>
<td>8.2</td>
<td>0.03</td>
</tr>
<tr>
<td>$P_2$</td>
<td>158</td>
<td>&lt;0.01</td>
<td>99</td>
<td>0.05</td>
</tr>
<tr>
<td>$P_3$</td>
<td>158</td>
<td>&lt;0.01</td>
<td>97</td>
<td>0.05</td>
</tr>
<tr>
<td>$P_4$</td>
<td>158</td>
<td>&lt;0.01</td>
<td>90</td>
<td>0.05</td>
</tr>
<tr>
<td>$P_5$</td>
<td>158</td>
<td>&lt;0.01</td>
<td>120</td>
<td>0.05</td>
</tr>
<tr>
<td>$P_{1,2,3,4}$</td>
<td>335</td>
<td>0.06</td>
<td>206</td>
<td>0.07</td>
</tr>
<tr>
<td>$P_{1,2,3,4,5}$</td>
<td>343</td>
<td>0.08</td>
<td>206</td>
<td>0.07</td>
</tr>
<tr>
<td>$P_1 \land P_2 \land P_3 \land P_4$</td>
<td>494</td>
<td>0.06</td>
<td>204</td>
<td>0.08</td>
</tr>
<tr>
<td>$P_1 \land P_2 \land P_3 \land P_4 \land P_5$</td>
<td>761</td>
<td>12.6</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

## Analysis

- More hardware (LEs) → Larger verification time, larger OH, more power req.
- However, order of magnitude smaller overheads than software-based enforcers
Conclusions

▶ As IMDs grow in complexity/connectivity they are increasingly vulnerable to attack.
▶ Run-time Enforcement can guarantee untrustworthy applications.
▶ Existing RE implementations not “secure” (they are usually software).
▶ Furthermore, implementations of Enforcers can themselves feature mistakes.
▶ We compile DTA policies to hardware-based enforcers.
▶ Hardware is intrinsically safer and more secure than complex software.
▶ The synthesized enforcers are automatically checked for correctness.
▶ Our enforcers guarantee a minimum safe QoS for IMDs.

Source code access

Source code for this project and its examples are available under the MIT license at https://github.com/PRETgroup/easy-rte


[21] S. Inc. (2017), Heartbleed bug,
