SAFE and Secure: Deeply Integrating Security in a New Hazard Analysis

Sam Procter (Software Engineering Institute)
Eugene Y Vasserman (Kansas State University)
John Hatcliff (Kansas State University)

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
Example Cyber-Physical System: Operating Room

Doctors

Nurses

Patient

Sensors

Computing Resources

Actuators
1. **Research Overview**
   1. Example Application
   2. Research Landscape

2. **Our Approach**

3. **Payoffs**

4. **Next Steps**
A Networked Medical Application

In the status quo, hospitals use “patient-controlled analgesia” pumps to manage temporary, severe pain.

Patients push a button to send a “bolus request” to trigger administration of a strong – typically opioid – analgesic.

Safety problems exist, but anesthesiologists have suggested integrating common sensors and simple application logic into a “closed loop” system.
A Networked Medical Application

Here's a conceptual view of what an integrated system might look like:

- **Sensors** – Measure respiratory health, i.e.: SpO\textsubscript{2}, respiratory and pulse rates, etc.

- **Controller** – Software to convert sensor readings into overall “health score” and enable / disable PCA pump

- **Actuator** – The PCA pump modifies the patient (i.e., controlled process) through responding (or not) to bolus requests
A Networked Medical Application

The system is (conceptually) simple, but exposes a number of complexities:

- **Heterogeneous Components**
  Different vendors may supply medical, networking, and computational components.

- **Variability** – The exact configuration isn’t known until the system is deployed and about to be used on a patient.

- **Network Enablement**
  Components expose behaviors, including actuation commands, over network interfaces.
Traditionally, safety of critical systems is analyzed using hazard analyses.

Most hazard analysis techniques pre-date modern levels of interconnectivity, though:

- FMEA (1949)
- FTA (1950s)
Traditionally, safety of critical systems is analyzed using *hazard analyses*.

Most hazard analysis techniques pre-date modern levels of interconnectivity, though:

- FMEA (1949)
- FTA (1950s)

Security has been integrated in some of these techniques (see Schmittner SAFECOMP16)

System Theory integrated recently (2011) by Leveson

- STPA-Sec and STPA-SafeSec followed

<table>
<thead>
<tr>
<th></th>
<th>Safety-Focused</th>
<th>Security-Aware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional Causality</td>
<td>FMEA, FTA, etc.</td>
<td>FMEA (Sec), FTA (Sec)</td>
</tr>
<tr>
<td>System Theoretic</td>
<td>STPA</td>
<td>STPA-Sec, STPA-SafeSec, SAFE</td>
</tr>
</tbody>
</table>
The Role of Guidewords

All of these techniques rely on the concept of **guidewords**.

- Terms used to guide analysis and ensure minimum coverage of basic concepts
- Almost always ad hoc, with no traceability to existing literature

Example guidewords from Leveson (2011)
Objective:
Unified set of semantic error/effect concepts, based on well-established literature, usable for safety and security.
1. Research Overview

2. Our Approach
   1. Dolev-Yao
   2. Worked Example

3. Payoffs

4. Next Steps
Systematic Analysis of Faults and Errors

Systematic Analysis of Faults and Errors (SAFE) is a new hazard analysis technique that works with STPA on technical elements of systems

• Importantly, SAFE can use any supplied set of guidewords

For this work, my co-authors and I wanted to look for a foundational basis for safety and security overlap.

We decided to start with guidewords for a number of reasons:

• Dictate failure modes considered by analysts
• Are intuitively understandable / don’t require extensive training
• SAFE’s configurability provides an excellent vehicle for testing them
A Dolev-Yao Based Guideword Set

Guidewords based on the Dolev-Yao model exhibit three desirable properties:

- Emphasize error *observability* – No mention of error cause
  
  - Separating cause from effect is a key part of SAFE & enables numerous benefits
- (Near) Completeness – Every* error type can be described
- (Near) Minimality – Removing a guideword from the set means that some errors cannot be described

<table>
<thead>
<tr>
<th>System Safety</th>
<th>Dolev-Yao</th>
<th>Network Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Read</td>
<td>Violate Privacy</td>
</tr>
<tr>
<td>Corrupt Value</td>
<td>Modify Existing</td>
<td>Craft Arbitrary Packets</td>
</tr>
<tr>
<td>Late / Dropped Message</td>
<td>Delay / Drop</td>
<td>Increased Latency / Packet Loss</td>
</tr>
<tr>
<td>Early Message</td>
<td>Craft and Send</td>
<td>Impersonate, Deny Service</td>
</tr>
</tbody>
</table>

* Barring pathological error behaviors
Worked Example – Methodology

We repeatedly re-performed part of SAFE’s original evaluation (on application logic) using different guideword sets.

We identified six improvements:

- **Alarms** – To alert a clinician of a problem that requires intervention
- **Timeouts** – To prevent message “flooding”
- **Timestamps** – To prevent delayed tickets from being used
- **Negative Ticket Values** – Specify “unsafe” time windows
- **Cryptographic Hashing/Signing** – To prevent message forgery
- **Encryption** – To prevent snooping on private medical data
Worked Example – Evaluation

Tried to judge how likely a guideword set was to suggest a potential design improvement

Dolev-Yao fared well, though sample size / subjectivity prohibit drawing firm conclusions

<table>
<thead>
<tr>
<th></th>
<th>Dolev-Yao</th>
<th>Avizienis Taxonomy</th>
<th>STPA-SafeSec</th>
<th>STPA/STPA-Sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alarms</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Timeouts</td>
<td>?</td>
<td>✔</td>
<td>?</td>
<td>✔</td>
</tr>
<tr>
<td>Timestamps</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Negative Ticket Values</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Cryptographic Hashing/Signing</td>
<td>✔</td>
<td>✖</td>
<td>✔</td>
<td>?</td>
</tr>
<tr>
<td>Encrypted Tickets</td>
<td>✔</td>
<td>✖</td>
<td>✖</td>
<td>✖</td>
</tr>
</tbody>
</table>

Legend: Will the guideword set suggest the design improvement?

✔ probably

? possibly

✖ probably not
1. Research Overview
2. Our Approach
3. Payoffs
   1. Effects Based Analysis
   2. Explicit Adversary Model
4. Next Steps
Effects Based Analysis

Merging Safety and Security

- Reduced Overhead – Less rework / duplicated analysis effort
- Fewer problems “fall through the cracks” – Many safety and security problems interact
- See Friedberg et al. in 2016 “Journal of Information Security and Applications”
Effects Based Analysis

Analysis Space Reduction

- Number of error causes are unbounded and may be unknowable
- Effects are (commonly) statically determinable and tightly bounded
- Similar to state-space reduction techniques in model checking

State space compression from Procter (2016)
Effects Based Analysis

Partial Independence / Compositionality

• Components can be analyzed independent of their input-producers
  - *(Not independent of output-consumers)*

Formal Methods

• Similar efforts in pure-software space to automatically derive assumptions required for safe operation
• See Rushby’s 2011 FACS “Assumption Synthesis”
Explicit Adversary Model Use

Dolev-Yao includes a complete set of network-based threats:

- But it excludes all others, e.g., compromised software/hardware (during development or at runtime).
- Threats that are excluded must be addressed (e.g., physical security, TPM chips, etc) and discharged as environmental assumptions.

Helpfully, all threats will necessarily manifest as a Dolev-Yao-classifiable error in successor components.
1. Research Overview
2. Our Approach
3. Payoffs
4. Next Steps
Future Work

Guidance on when certain guideword sets are appropriate
- Sourced from academic, industrial, and governmental/regulatory authorities
- Deeply integrated with tooling and techniques [Procter and Hatcliff, ASSURE15]

Continue to move system safety and formal methods communities closer
SAFE and Secure: Deeply Integrating Security in a New Hazard Analysis

Sam Procter (sprocter@sei.cmu.edu)
Eugene Vasserman
John Hatcliff

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213