Survivable Software for Safety-Critical Cyber-Physical Systems

Karen Mercedes Goertzel, CISSP
Lead Associate
703.698.7454
goertzel_karen@bah.com

https://www.surveymonkey.com/s/ASDC2012_Talk40

Background: Cyber Security

- Primary concern: threats to security of information accessible on a network (e.g., the Internet)
- Secondary concern: threats to systems (including networks and applications) that create, transmit, process, and store that information
- Threats to cyber systems and the information they handle are:
  - Realized as misuse/abuse (exploits and attacks)
  - Intentional: direct human causation, with either direct or indirect (proxied by malware) human involvement
  - Mainly deterministic (non-stochastic)
    - Initialization, timing, and location may be unpredictable
    - Objectives, targets, and attack patterns become increasingly predictable as attack progresses
    - “Randomness” is usually pseudo, so ultimately predictable
    - Patterns can ultimately be discerned even in Advanced Persistent Threats

Background: Cyber Security Threats (cont'd)

- Complex
  - Multiple exploit and attack events may be localized, distributed (horizontally and vertically), or both
  - Combines direct and non-direct attacks
  - Combines automated, semi-automated, and manual exploits and attacks
  - Causes byzantine, non-contiguous faults and multiple failures

- Often persistent: new strategies supplant or augment failed ones

- Consequences of realized threats
  - Vary according to the nature and sensitivity of the information, and its criticality to the business function or mission that uses it
  - Usually able to be characterized in terms of financial loss

Background: Cyber Security (cont'd)

- Goal of cyber security:

  To prevent the cyber system from becoming the means by which the information created, processed, transmitted, or stored by that system can be inappropriately disclosed, modified, or destroyed

  - This goal is achieved by:
    - Protecting systems and information against realized
    - Detecting indications of realized threats
    - Reacting to verified threats
    - Recovering from successfully realized threats

Background: Safety-Critical Systems

- Main concern: hazards
  - Realized as mishaps (causing software or hardware faults and failures)
  - Unintentional causation: human error, accident, or “act of god”
  - Partly or wholly non-deterministic (stochastic): occurrence may be unpredictable, but outcome is generally predictable
  - Straightforward: single, localized event leads to simple fault or failure

Background: Safety-Critical Systems (cont'd)

- Potential consequences if realized:
  - Physical injury or death
  - Destruction of the means of survival (food, water, shelter, medicine)
  - Environmental damage, with potential to damage health

  - Destruction of means of financial livelihood or support
  - Very difficult to characterize in terms of actual (vs. arbitrarily-defined/actuarial) financial loss

Cyber-physical systems

- "Touch" the physical world in some way
- Main functions: Monitoring, diagnosis, and control of physical processes
- Subject to safety hazards (because they're physical) and security threats (because they're cyber)
  - Need to avoid hazards and tolerate faults drives the imperative for safety
  - Need to protect against threats and tolerate intrusions drives the imperative for security
- Can be embedded, non-embedded, or a hybrid of the two

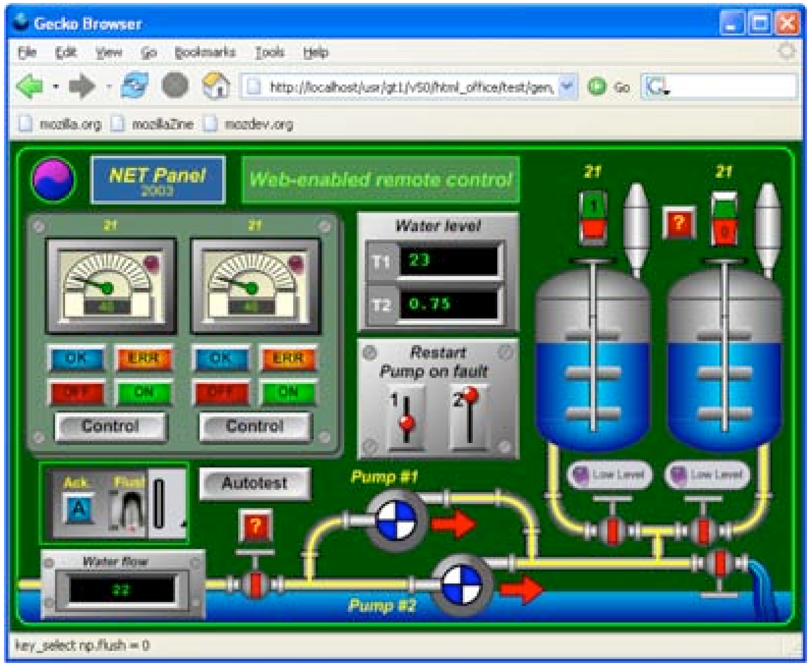
Cyber-physical systems (examples)

- **Embedded**
  - Vehicle controllers/computers (accessible via vehicle telematics, e.g., GM OnStar)
  - Network-connected (incl. RFID networks) medical devices
  - Embedded components of industrial and infrastructure control systems
    - process controllers*
    - intelligent electronic devices (IEDs)
    - remote telemetry units (RTUs)
    - sensors
  - Remote-controlled robots
  - Weapon systems
  - Unmanned aerial vehicles

**programmable logic controllers often used*

Non-embedded cyber-physical systems (examples)

- **Embedded (cont’d)**
  - – Maritime and avionic navigation devices
  - – “Smart” appliances and “smart” houses (accessible via Internet or phone network)
- **Non-Embedded**
  - – Air traffic control systems
  - – Avionic and space flight control systems
  - – Non-embedded components of industrial and infrastructure control systems
    - Supervisory Control And Data Acquisition (SCADA) systems
    - Distributed control systems (DCS)
  - – SmartGrid
  - – Railway signalling and track switching systems



Recorded cases of safety-critical and cyber-physical systems "sabotaged" by software errors

- Medical devices (acc. to FDA, 33% of all recalls due to software failures)
- Weapon systems
- Military planes and helicopters
- Telephone switches
- Space vehicles
- Soviet early warning system
- Automobile recalls
- Environmental monitoring systems
- Prisoner monitoring system
- Industrial Control Systems (buffer overflows, heap overflows, memory corruption vulnerabilities in multiple SCADA products in U.S., Europe, and China)

Critical infrastructure at risk

- Industrial control systems combine data processing, command and control, and physical process control
  - Use open networking protocols, remote access, even Internet connectivity
  - Combine old legacy software never meint for networked use with new components on mobile devices using wireless for remote access/control
  - Numerous vulnerabilities at system and software levels exposed to insider and outsider threats
  - Outcomes of threats and hazards are so similar, the root cause may be impossible to determine
    - *Example:* Systematic sabotage of Maroochy Shire (Australia) waste water treatment plant by Vitek Boden, ex-IT consultant, using stolen SCADA software, laptop, and WiFi.
      - Most of Boden's 46 separate "break-ins" over 4 months were mistaken for stochastic failures
      - Outcome was the same: Millions of litres of raw sewage released into surrounding rivers, resort grounds, and nature reserves

Security of safety-critical cyber-physical systems

- Safety ***always*** trumps security
  - Safety remains the #1 imperative in safety-critical cyber-physical systems
  - Security is only the means to the end of guaranteeing safety
  - Security that does not guarantee safety has no value
  - Security must NEVER impede safety
- Security threats represent a new, unprecendented type of “intentional hazard”
  - Safety engineering takes physical into account, but not cyber
  - Safety requirements are based on models of hazards, faults, and errors, not threats and attacks
  - Events typical of cyber attacks are not captured in most hazard models
  - No distinction between unintentional and intentional causality

Increasing vulnerability and exposure

- Growing dependence on software alone
  - No manual/mechanical backup in case of failure
- Increasing use of commodity/open source components
  - Larger, more complex, harder to analyze, more dormant code
  - Unknown pedigree/provenance (supply chain issues)
  - Prone to more flaws, including those exploitable as vulnerabilities
  - Many components used are general-purpose, e.g., Windows, never engineered for safety-critical and cyber-physical stresses
- Increased connectivity via exposed networks
  - Leveraging the Internet
  - Leveraging wireless (cell, satellite, RFID)
  - Embedded systems often too “lite” to implement full range of network security protections (cryptographic and non-cryptographic)

Intentional sabotage by SDLC "insiders"

- Canadian gas pipeline control software sabotaged through time/logic bomb Trojan
  - CIA sabotaged then allowed software to be stolen by USSR
  - Once installed on Trans-Siberian Pipeline, time bomb "went off": reset pump speeds and valve settings
  - Result: massive pressure overload caused ruptures in pipeline joints and welds
  - Resulting explosion equivalent in power to that of 3-kiloton nuclear weapon

Intentional sabotage post-SDLC

- Stuxnet
  - First known programmable logic controller rootkit and industrial control system (ICS) spyware
  - Exploits software tools to modify and download code changes to programmable logic controllers on specific Siemens ICS
  - Able to replicate with or without network (e.g., via USB sticks and jump drives)
  - Too sophisticated to be work of individual hacker(s); probably nation-state sponsored (Israel? U.S.? Both?) and aimed at Iranian nuclear plants
  - Spread initially thanks to Windows vulnerabilities
  - Exploited zero-day vulnerabilities specific to the Siemens ICS software

Research proof-of-concept exploits

- 2011: Luigi Auriemma publishes (on Bugtraq) proof-of-concept attack code that exploits 34 software vulnerabilities in five products from four leading SCADA vendors
- 2011: Univ. of California-San Diego and Univ. of Washington researchers demonstrate cellphone based attack on automobile telematic system made possible by reverse engineering telematics protocol implementation and exploiting buffer overfow vulnerabilities in automobile’s software modem.
  - Attack enabled them to compromise several onboard vehicle systems
  - Attackers gained control of engine, brakes, lights, instruments, radio, and locking system
- 2008: Medtronic combination defibrillator/pacemaker was hacked by researchers using $30K of lab equipment, who were able to reprogram the compromised device to shut down intermittently, and to deliver potentially-fatal levels of electricity

What influences software safety and security

- *SDLC principles and practices:* The practices used to develop the software, and the principles that governed its development, testing, distribution, deployment, and sustainment;
- *Development tools:* The programming language(s), libraries, and development tools used to design, implement, and test the software, and how they were used by the developers;
- *Third-party components:* How COTS, open source, legacy, and other third-party-developed components were evaluated and chosen, tested, constrained and integrated;
- *Runtime configuration:* How the software was configured upon installation and execution;
- *Execution environment:* The software's access to required environment-level resources and services; the exposure to hazards and threats of higher-level software by its execution environment;
- *Practitioner knowledge:* Security and safety knowledge of the software's analysts, designers, developers, testers, and maintainers…or their lack thereof.
- *Size, complexity, autonomy:* Of the software code (at rest and in execution)

What constitutes the SDLC for survivable software?

- Safety engineering and secure development principles and practices
- Adequate requirements, architecture, and design that
  - Reflect correct developer assumptions
  - Form an adequate basis for implementing software that will
    - Operate in a dependable and trustworthy manner
    - Use only appropriate, reliable, secure interfaces to external components/services, administrator consoles, users/clients
    - Anticipate state changes and failures that could result from accidental and intentional errors and faults
  - Address safety and security concerns associated with third-party components
- Safe and secure coding practices and supporting tools
- Safety and security verifications and validations at all SDLC checkpoints (requirements, architecture, design, and code reviews; unit, integration, deployment, and regression testing.)

What constitutes the SDLC for survivable software? cont’d

- Safe, secure assembly/integration of components that
  - Ensures that all interfaces and are inherently safe and secure or have necessary safety and security constraint mechanisms added
  - Minimizes of high-risk and known-vulnerable components’ exposure at the software boundary
- Safety and security analyzes and tests (e.g., static and dynamic code analyses, fault injection, fuzzing, penetration testing)
- Secure distribution and deployment
  - “Sanitization” of software executable(s) to remove all unsafe and exploitable backdoors, data, etc.
  - Distribution via media or channels that adequately protect the software from tampering, with mechanisms to verify received software’s integrity
  - Default configuration settings are maximally constrained/restrictive

What constitutes the SDLC for survivable software? cont'd

- *Safety and security in sustainment:* Maintenance with safety and security in regression tests; hazard and vulnerability management, with timely issuance of fully tested, impact-analyzed patches and updates
- Development, testing, and deployment tools that support/enhance and do not degrade safety and security
- Secure configuration management systems and processes
- Safety and security-knowledgeable developers, testers, installers
- Upper management commitment to safety and security

Safety engineering

- Impressively scientific and disciplined
- *Objective:* extreme reliability despite presence of stochastic hazards
- *Techniques*
  - software quality techniques
  - specification based on careful and thorough hazard analyses
  - fault-tolerant design
  - extensive testing
  - safe subsets of programming languages
  - formal methods for specification, modeling, verification
- Little or no consideration of intentional misuse/abuse (vs. mishaps) or threats (vs. hazards)

Security of safety-critical software

1. Functional (including usage) and non-functional (interface, data, constraint, and quality) requirements
   - Need to capture requirements and test cases based on use, misuse, and abuse cases, threat/attack models and hazard/mishap models
2. Deployed executable
   - Need to assure deployed executable’s integrity
   - Need to assure executing software’s availability
3. Data created, stored, transmitted, or used by the software
   - Need to assure confidentiality, integrity, and availability of inputs and outputs
4. Software’s interfaces and execution environment
   - Need to assure availability, integrity, and accessibility of environment (hardware, software, network) resources and services relied on by the software
   - Need to assure access to the deployed software by authorized users and only authorized users (human and other software)

Examples of Safety vs. Security Requirements

1. When in mode *V*, the system shall not cause accidental harm of type *W* to valuable assets of type *X* at an average rate > asset value *Y* per *Z* time duration.
2. When in mode *W*, the system shall not cause mishaps of type *X* with an average rate of more than *Y* mishaps per *Z* trips.
3. When in mode *X*, the system shall not cause hazard *Y* to exist more than an average of *Z* percent of the time.
4. Upon detecting an accident of type *W* when in mode *X*, the system shall react by performing function *Y* an average of at least *Z* percent of the time.

1. When in mode *V*, the system shall limit the occurrence of malicious harm of type *W* to valuable assets of type *X* at an average rate > asset value *Y* per *Z* time duration.
2. When in mode *W*, the system shall prevent the first successful attacks of type *X* for a minimum of *Z* time duration.
3. When in mode *X*, the system shall not have security vulnerability *Y* for more than an average of *Z* percent of the time.
4. Upon detecting a misuse of type *W* when in mode *X*, the system shall react by performing function *Y* an average of at least *Z* percent of the time.

Source: Donald Firesmith, Carnegie Mellon University Software Engineering Institute

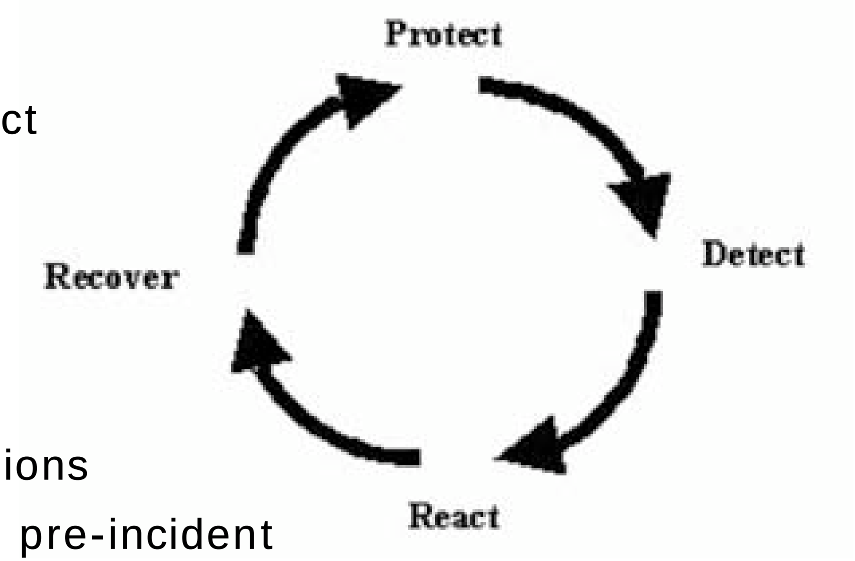
Security of safety-critical software (cont'd)

System vs. software level security

- *System level:* Focus is on
  - external interfaces and interactions (calls/requests and inputs/outputs)
    - between system components
    - between system and other systems
    - between system and users
- *Software level:* Focus is on
  - internal workings *and*
  - external interfaces and interactions (calls/requests and inputs/outputs)
    - between software components/units/modules
    - between software and execution environment
    - between software and users or process

Detect-protect-react-recover

- *Detect*
  - log (events)
  - audit (usage)
  - sense (anomalies, intrusions)
  - monitor (execution, interactions [inputs/outputs])
- *Protect* through “defense in depth” and “defense in breadth”
- *React*, in hopes of
  - minimizing extent, intensity, duration of impact
  - minimizing likelihood of recurrence by
    - blocking certain types of inputs
    - terminating user sessions
    - terminating some or all functionality
    - disconnecting some or all network connections
    - assessing damage, attempting recovery to pre-incident state



Moving from *detect-protect-react-recover* to *survive*

- Safety-critical cyber-physical systems have no tolerance for delays associated with post-incident recovery
- Need ability to tolerate attacks and survive threats in the same way they tolerate faults and survive hazards
- Need ability to survive even persistent, high-intensity intrusions and attacks

Cyber-physical system survivability

- System level
  - *Redundancy:* hot sparing, decoupling and replication high-consequence components
  - *Rapid recovery:* automatic backups, automatic swap-over of high-consequence components

Cyber-physical system survivability

- Software level
  - Diversity, redundancy, fault-tolerance
  - Exception-handling logic that
    - is purpose-written, not generic
    - typically exceeds quantity of associated program logic
    - minimizes ability of faults to aggregate/escalate into failures
    - enables continued operation of critical functions rather than failure
      - graceful degradation: lower priority functions terminate while critical functions keep running
      - Prevents software from simply crashing, entering non-secure state, or “dumping core” (including cache, temp files)

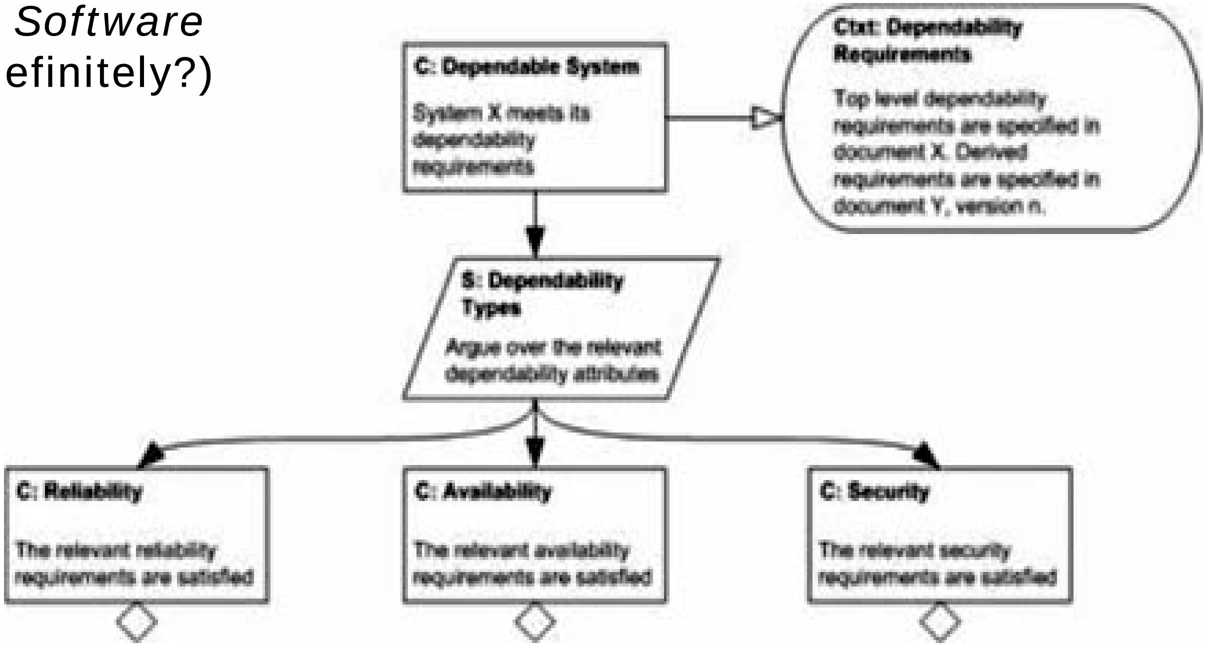
Software engineering for survivability
Need simpler formal methods

- “Semi-formal” and “lightweight” formal methods, usually tool-supported, enable wider adoption outside traditional specialty software engineers
  - *Examples:* Praxis High Integrity Systems’ Correctness-by-Construction, Object Management Group’s Model Driven Architecture
- Automation of formal methods for non-expert use
  - *Examples:* Munich University of Technology’s Autofocus and Quest, Jean-Raymond Abrial’s B-Method, Jan Jürjens’ UMLSec and UMLSafe

Booz | Allen | Hamilton

Software engineering for survivability
Need hybrid safety / security assurance cases

- Approaches available now or soon
  - Praxis High Integrity Systems SafSec (developed for UK MOD)
  - ISO/IEC 15026, *System and Software Engineering—System and Software Assurance* (postponed indefinitely?)


C: Dependable System

System X meets its dependability requirements

Ctxt: Dependability Requirements

Top level dependability requirements are specified in document X. Derived requirements are specified in document Y, version n.

S: Dependability Types

Argue over the relevant dependability attributes

C: Reliability

The relevant reliability requirements are satisfied

C: Availability

The relevant availability requirements are satisfied

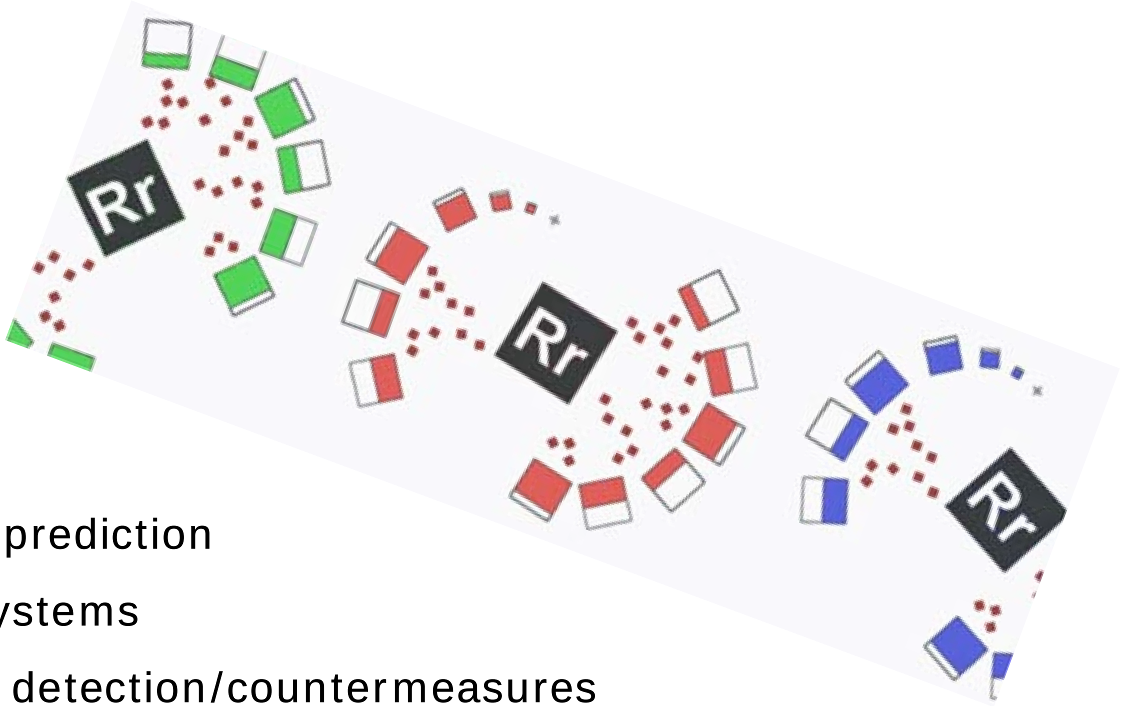
C: Security

The relevant security requirements are satisfied

Software engineering for survivability

Need to use emerging diversity + redundancy techniques

- polymorphism (same techniques used by virus/worm developers)
- dynamic software composition
- *n*-version programming
- code filtering
- pseudo-genetic transformations of executable code over time
- software rejuvenation
- software evolution
- phylogenetic trees for vulnerability prediction
- nature-based models of software systems
- computer immunology for anomaly detection/countermeasures



To date: numerous promising prototypes, few if any robust real-world implementations

Software engineering for survivability
Need to establish the discipline of "survivability engineering"

- *Goals:* investigate, develop, and prove effectiveness of engineering techniques and tools that produce survivable software, and assure software's survivability with regard to both safety hazards and security threats
- *Examples of research*
  - Carnegie Mellon Software Engineering Institute: Survivable Systems Engineering
  - University of Virginia Dependability Research Group: Willow system
  - National Center for Supercomputing Applications (NCSA): Mithril
  - Stony Brook University: Survivable Software project
  - INRIA: Project Gallium

Software engineering for survivability
SDLC process adaptations

- Augment hazard analyses with threat models
- Add security properties to formal specifications/models
- Adopt secure design principles and practices (least privilege, separation of roles/duties/and domains, etc.)
- Add security functions to safety-critical system designs
- Augment safe coding practices, standards, tools with secure coding practices, standards, tools
- Add security "faults" to safety faults generated for fault injection testing
- Add security analysis and testing techniques and tools to current test regime (e.g., static analysis for security, fuzz testing, vulnerability scans, penetration tests)
- Develop and validate hybrid safety/security assurance cases
- Consider adopting safety/security convergence methodologies, process models, e.g., Correctness by Construction, FAA/DoD Safety and Security Extensions to iCMM/CMMI

Recap

- No system is safe if that safety can be intentionally sabotaged
- Security threats are growing in complexity, intensity, proliferation, and unpredictability
- Detect-protect-react-recover paradigm is inadequate to assure security of safety-critical systems
- Safety-critical cyber-physical software is vulnerable and exposed
  - commodity components and technologies
  - networking and remote accessibility
  - dependence on pure software control with no manual/mechanical backup
- At risk: not just critical infrastructure systems (SCADA, etc.), but weapons systems, avionic systems, automotive systems, air traffic control systems, medical systems, etc.

Safety-critical cyber-physical systems *must* be able to *survive* both mishaps and misuse.

Suggested Resources

- Naval Ordnance Safety and Security Activity *Safety and Security Considerations for Component-Based Engineering of Software-Intensive Systems*

  https://buildsecurityin.us-cert.gov/swa/downloads/NAVSEA-Composition-DRAFT-061110.pdf

- DHS/DACS *Enhancing the Development Life Cycle to Produce Secure Software*

  https://www.thedacs.com/techs/enhanced_life_cycles/

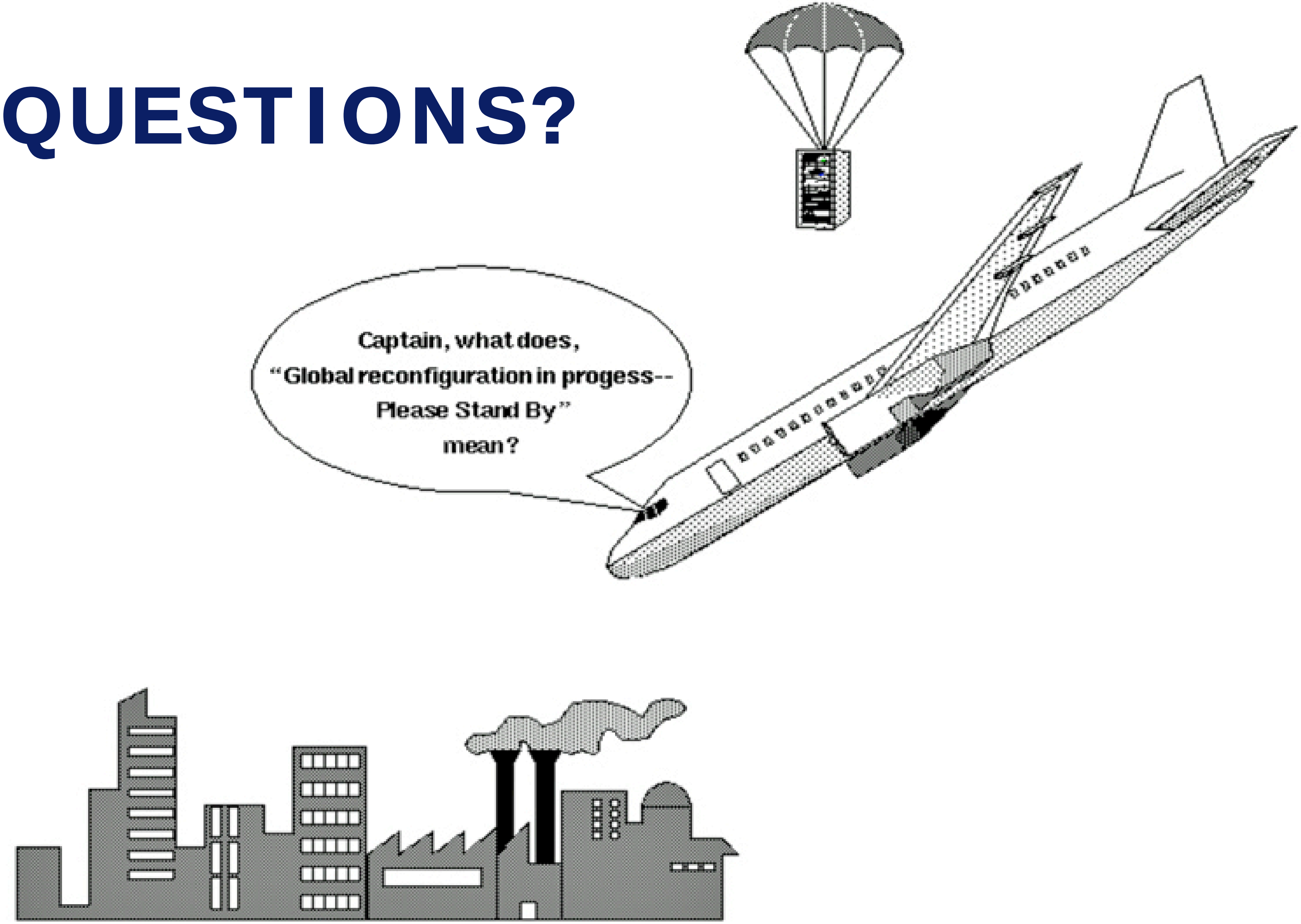
- NATO *Engineering Methods and Tools for Software Safety and Security*

  http://www.booksonline.iospress.nl/Content/View.aspx?piid=11975

- Jeffrey Walton “Protection Schemes Based on Virus Survival Techniques”

  http://www.codeproject.com/Articles/21081/Protection-Schemes-Based-on-Virus-Survival-Techniq

QUESTIONS?



https://www.surveymonkey.com/s/ASDC2012_Talk40