Software survivability: where safety and security converge

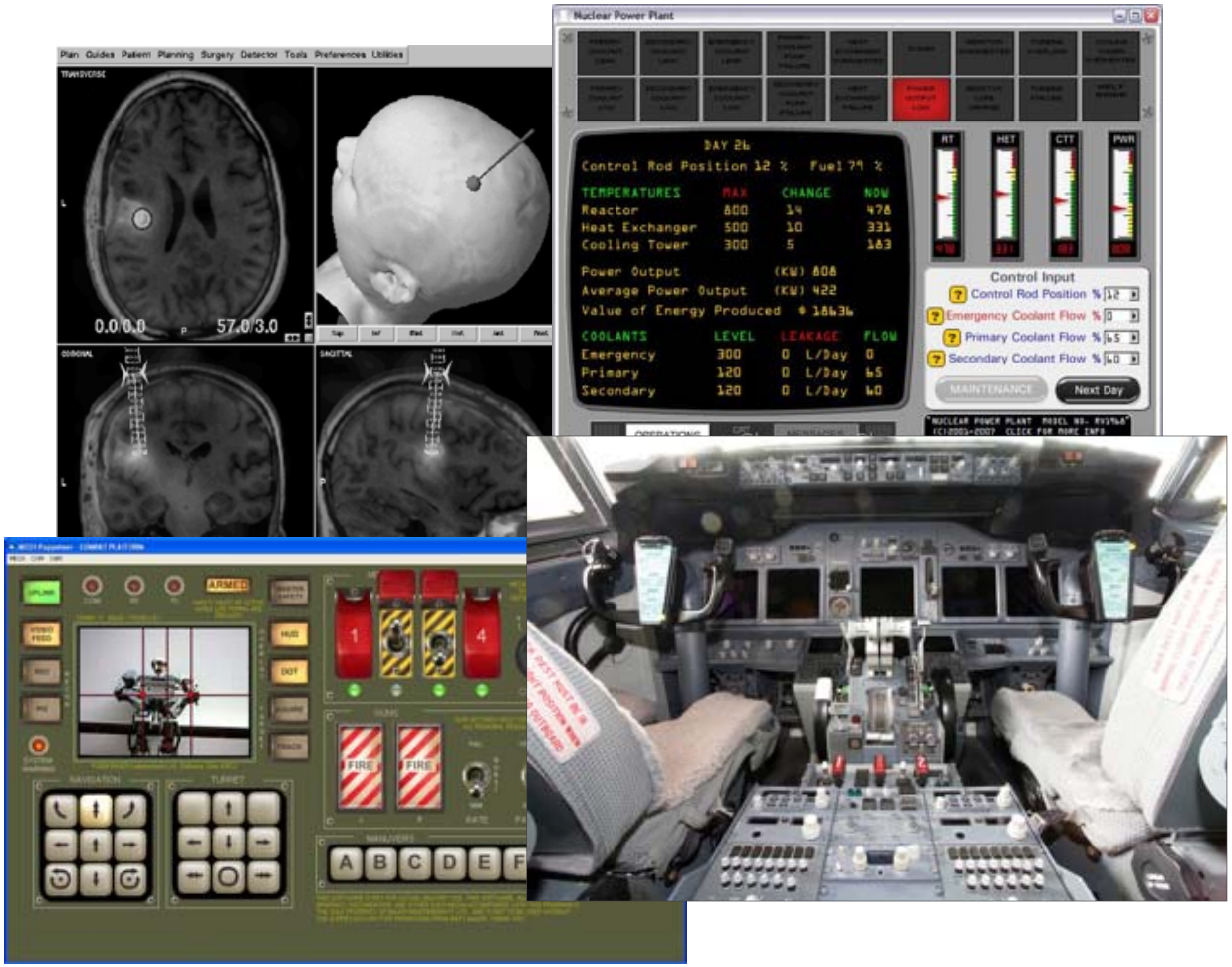
KAREN MERCEDES GOERTZEL, CISSP
goertzel_karen@bah.com 703.698.7454

Safe and Secure Systems and Software Symposium (S5)
Beavercreek, OH—15 June 2010

Booz | Allen | Hamilton

The Challenge:
Safety-critical systems are increasingly…

- software-intensive/software-reliant (no manual or mecha-nical backup)
- built from “non-developmental” components (including COTS)
- network access
- susceptible to malware, cyber attacks, supply chain attacks



Safety-critical systems

- **Safety:** "Software System Safety implies that the software will execute within a system context without contributing to hazards."
  —Leveson
  - *Hazards:* Conditions that can cause death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment*—MIL-STD-882D*
- **Main functions:** monitoring, diagnosis, and/or control of *physical* systems
- **Consequences of failure:** potentially catastrophic, even fatal

Security-critical systems

- **Security:** To be considered secure, software must exhibit three properties: *dependability*, *trustworthiness*, and *survivability*
  —Enhancing the Development Life Cycle to Produce Secure Software
- **Main functions:** protecting information and the systems and networks that process and transmit information against attacks
  - Involves detecting, responding to, and recovering from attacks
- **Consequences of failure:** depends on purpose, sensitivity, and criticality of the information, system, network

Examples of security-critical systems

- **Non-embedded:**
  - operating system kernels, file systems, database management systems
  - virtual machine monitors/hypervisors
  - enterprise security management systems, security monitoring and management systems
  - single sign-on servers
  - antimalware, antispyware
  - data leakage detection/prevention
  - network-based and host-based agents and sensors
  - public key infrastructure, X.509 certificate handling, digital signature, SAML, and XACML applications, etc.
- **Embedded:**
  - facility security sensors, alarm systems
  - physical access control systems
  - Trusted Platform Modules (TPMs), Hardware Security Modules (HSMs)
  - “security appliances” (firewalls, intrusion prevention/detection systems, secure routers/network controllers)
  - cryptographic devices
  - smart cards, password tokens

Hazards

- Stochastic (accidental)
- Usually straightforward
- Often single, localized
- Occurrence is unpredictable, but outcome is generally predictable
- Result: simple faults

Threats

- “Non-stochastic” (intentional), or stochastic (unintentional) with intentionally exploitable results
- Often complex
- Multiple, coordinated but often diverse, non-contiguous/ distributed “hits”
- Occurrence often unpredictable (may become more predictable over time); outcome may or may not be predictable
- Require human intention and intelligence in planning and execution
- Result: byzantine faults

Attacks on safety-critical systems can have catastrophic results

- Unintentional weaknesses and errors have proved fatal
  - *Examples:* Therac-25, Ariane 5 Flight 501, Toyota Prius
- Intentional threats target/exploit unintentional weaknesses and faults
- Catastrophes arise due to software/physical boundary
  - Example: CIA sabotage of Trans-Siberian gas pipeline
  - Logic bomb written by CIA added to embedded software in pipeline controller illegally exported to USSR by Canadian firm working with CIA

Security of safety-critical infrastructure

- Hybrids of information, command and control, and physical process control systems
  - Support open networking protocols, remote access, even Internet connectivity
  - Increasingly include components hosted on mobile devices, using wireless communications
    - *All this introduces exploitable vulnerabilities*
  - Outcomes are similar for hazards and threats, making true cause hard to diagnosis

- Example: Maroochy Shire (Queensland, Australia) wastewater treatment system sabotage by disgruntled former software contractor

Proliferation of attacks on safety-critical systems due to increased opportunity

- Increased “software intensiveness” of everything: software ubiquity; functions formerly performed by hardware, now mainly or wholly done by software
- Systems (even highly critical embedded systems) built from larger, more complex and vulnerable commodity/open source components)
  - Attackers have deep knowledge of the products and technologies used…and their exploitable vulnerabilities.
- Exposure on publicly-accessible networks, including wireless (cellular, SATCOM, RFID)
- Embedded no longer means isolated (OnStar, remote-controlled medical devices/robotics)

Security-critical system attacks can also have catastrophic results

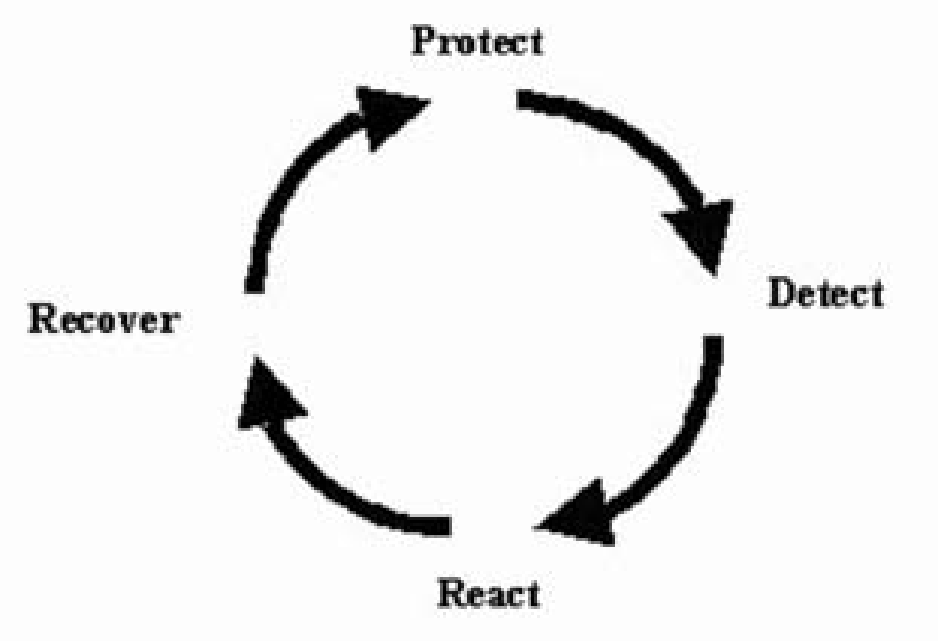
- **Top secret intelligence database hack:** Names of U.S. agents operating in Moscow stolen for sale to USSR (Robert Hansen)
- **Command and control system hack:** Secret planned troop deployments disclosed to the enemy, enabling preemptive attacks
- **Logistics system hack:** Troop quantities, destinations, transport dates/vessels disclosed, abetting enemy targeting
- **Major banking system hack:** Millions of dollars undetectably siphoned from accounts a few dollars at a time
- **Electronic voting system hack:** "Fixes" the election
- **Facility security system hack:** Price-less, irreplaceable artworks stolen

Safety-critical and security-critical systems: convergences and divergences

- **Convergence:** Both types of systems need to continue operating dependably under extraordinary conditions
- **Divergences:**
  - What constitutes an “extraordinary condition”—hazard vs. threat
  - What is at stake if the software fails

Security: traditional view

- Detect
  - log (events)
  - audit (usage)
  - sense (anomalies, intrusions)
  - monitor (execution, I/O)
- Protect
  - “defense in depth”
  - “defense in breadth”
- React and recover
  - minimise impact: extent, intensity, duration
  - minimise likelihood of recurrence
    - *block certain types of inputs/outputs*
    - *terminate user sessions*
    - *terminate some or all functions*
    - *drop some or all network connections*
    - *assess damage, attempt recovery to pre-incident state (roll-back)*



Today's infowar cyberadversaries

- **Knowledgeable:** They know more about our software than we do, including its vulnerabilities (thanks to National Vulnerability Database, Common Vulnerabilities and Exposures, etc.)
- **Skill and sophisticated:** Not just "script kiddies". Attackers know how to exploit vulnerabilities, how to augment/assist direct attacks with social engineering and surreptitious malware (worms, Trojans, bots, spyware)
- **Quick:** "Zero day" is the rule, with new attacks appearing before vulnerabilities are discovered by developers, let alone patched
- **Motivated and well-resourced:** Not just recreational hackers, but organized criminals, nation-state infowarriors, cyberterrorists

Traditional security can be just plain wrong for safety-critical systems

- Traditional security relies on service interruptions to prevent incident escalation or recurrence
  - Safety-critical systems need 100% uptime
- Traditional security requires tolerance for delays associated with post-incident recovery
  - Safety-critical systems must operate within stringent performance bounds and thresholds

As fought today, information wars are not just being lost, they cannot be won.

- The answer: ***Survivability***
  - The ability to “fight through” high-intensity attacks without service interruptions or delays

Safety-impacting security properties

- **Integrity** *(intactness)*
  - *Threats:* tampering, corruption, subversion, augmentation
- **Availability** *(presence, accessibility, timeliness)*
  - *Threats:* denial of service, sabotage, deletion, interception/rerouting or hijacking
- **Authenticity** *(not spoofed or substituted)*
  - *Threats:* counterfeiting (creates false assumptions, can reduce quality and threaten integrity)
- **Confidentiality** *(secrecy of design and implementation details)*
  - *Threats:* reverse engineering (to learn exploitable vulnerabilities, better craft attacks, bypasses, countermeasures)

Safety engineering: impressively scientific and disciplined

- Software quality engineering as baseline foundation: error-free, correct, predictable execution
- Specifications based on careful and thorough models, simulations, hazard analyses
- Formal methods verify modeled requirements and high-level design correctness with mathematical precision
- Fault-tolerance features, especially at system level
- Code programmed using safe language subsets (e.g., MISRA-C, SPARKAda)
- Extensive testing (source and binary levels) to verify safe behavior in presence of hazards
  - *BUT*…little or no consideration of malicious events

Security of safety-critical software must be addressed at three levels

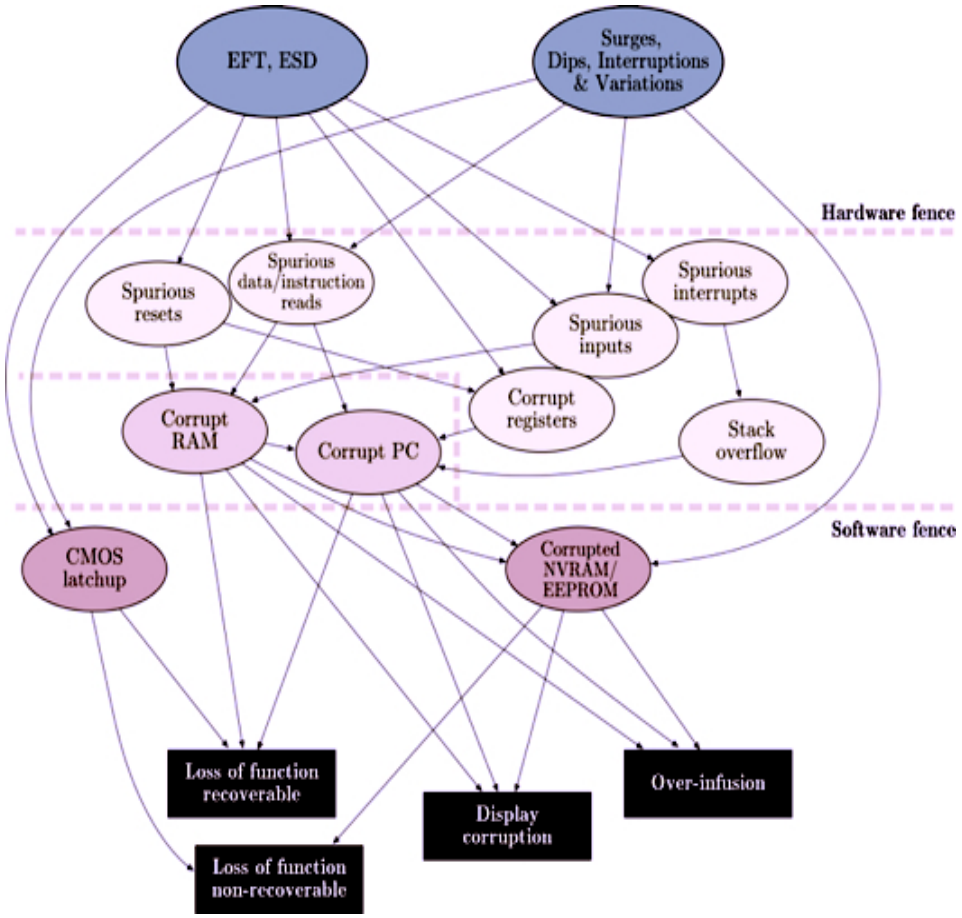
- **Functional:**
  - threats to software's availability and integrity
- **Data:**
  - threats to integrity, availability, confidentiality of inputs, outputs
  - threats to integrity, availability, confidentiality of data being processed, stored, transmitted
- **Execution environment:**
  - threats to availability, integrity of environment components
  - threat of resource theft

Software safety and security: how software reacts to input/stimuli from external sources

- Intercomponent interfaces (APIs, RPCs)
- External interfaces with execution environment (APIs, system calls)
- Hardware/software boundary events (sensor inputs—generated by measurement or changes in physical environment, e.g., temperature, velocity, pressure, altitude, decibels, etc.)
- External interfaces with other systems (message passing)
- External interfaces with humans (direct inputs/stimuli)

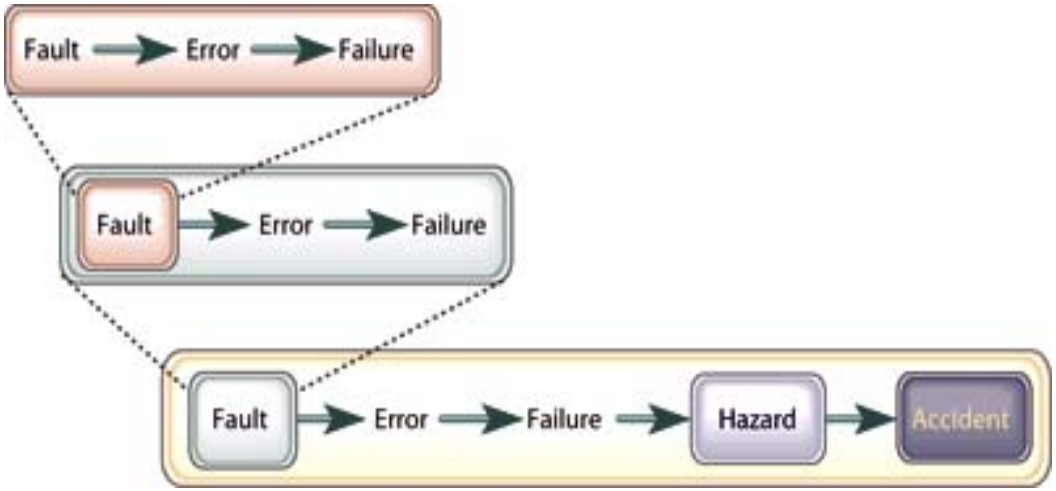
Hazards and risks that arise in software

- Parameter-passing issues
- Timing and sequencing issues
- Resource conflicts/deadlocks
- Improper configuration
- Improper, inadequate, and unnecessary functionality
- Unanticipated execution of unused logic/dormant code
- Input validation and data protection issues
- Inherent module/component weaknesses
- Inherent architecture/frame-work weaknesses



Modeling and analysis of hazards and risks

- Hazard analysis
  - FMEA, FMECA, iFMECA
  - formal models/proofs
  - cause/consequence analysis
  - fault tree analysis
- Threat modeling
  - STRIDE/DREAD, etc.
  - attack trees/graphs
- Architectural modeling, tradeoff analysis
- System-level analyses (modeling, simulation)

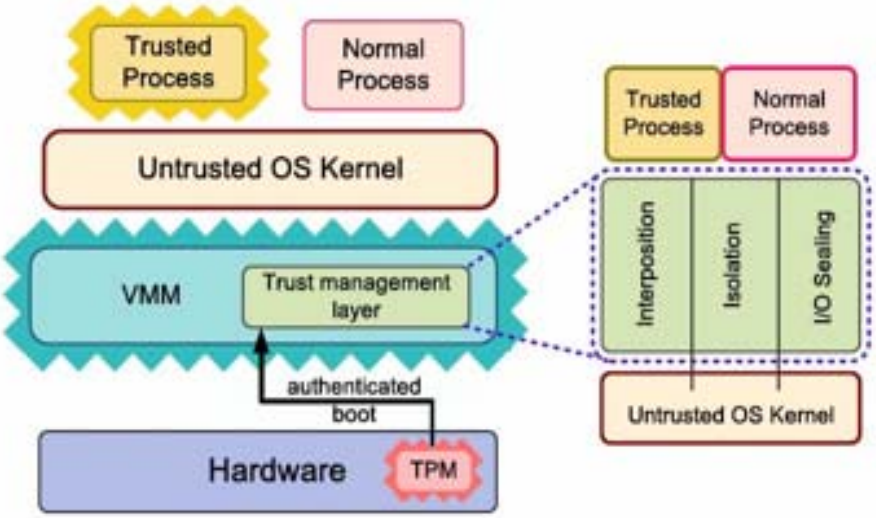


Properties of survivable software systems

- Minimal exposure of untrustworthy and vulnerable components
- Only known-trustworthy components perform safety- and security-critical roles
- Trustworthy components isolated from untrustworthy components
- Constraint of damage from component misbehaviors
- Survivability techniques leveraged for tolerance of both stochastic and *non*-stochastic faults
- Flexible architecture allows interpolation of controls and countermeasures between modules/components, and between system and external entities
- Abstraction of component specifications mean specific components can be replaced *at any time* by functionally-comparable component(s), to mitigate hazards/risks
- Extensive analysis/testing considers hazards *and* threats

Mitigations for unsafe, non-secure software

- **Extension of existing software** *(rewriting, wrapping, APIs)*:
  - *Functional logic:* e.g., specific, explicit error and exception handlers
  - *Interface logic:* validation and filtering of unacceptable/dangerous inputs or outputs
  - *Safety/security controls and countermeasures*
- **Architecture-level controls and countermeasures** *(proxies, filters, gateways, execution environment constraints)*:
  - external safety and security checks/monitors and interlocks
  - VMs, TPMs, HSMs
  - fault tolerance measures



- **Substitute/replace unacceptable software:** Use known-trustworthy, vetted alternative(s), or custom-develop

Safety and security verification and validation of implemented software

- Property-based testing (formal method)
- Source code fault injection with fault propagation analysis
- Binary fault injection
- Fuzzing
- Source code static and dynamic analyses
- Binary executable analysis (e.g., function extraction, reverse engineering, binary scans)
- Functional tests: individual component, pair-wise, full system levels
- Vulnerability scans
- Penetration tests

Software engineering trends aid survivability

- Survivability engineering techniques and tools expressly intended to produce and V&V survivable software
- New redundancy and diversity techniques based on techniques used by nature, and by malware developers
- Tool-supported, automated formal, “semi-formal”, and “lightweight” formal methods for wider adoption by non-experts
- Safety and security extensions to SDLC processes and methods
- Hybrid safety/security assurance cases

Adapting the SDLC for safe *and* secure software

- 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/domains, etc.)
- Add security functions to safety-critical system specs, designs
- Augment safe coding practices, standards, tools with secure practices, standards, tools
- Add security faults to those generated for fault injection testing
- Add security analysis and test techniques and tools to current test regime (e.g., code security review, vulnerability scans, pen tests)
- Develop and validate hybrid safety/security assurance cases

Minimum acceptable practice for survivable software

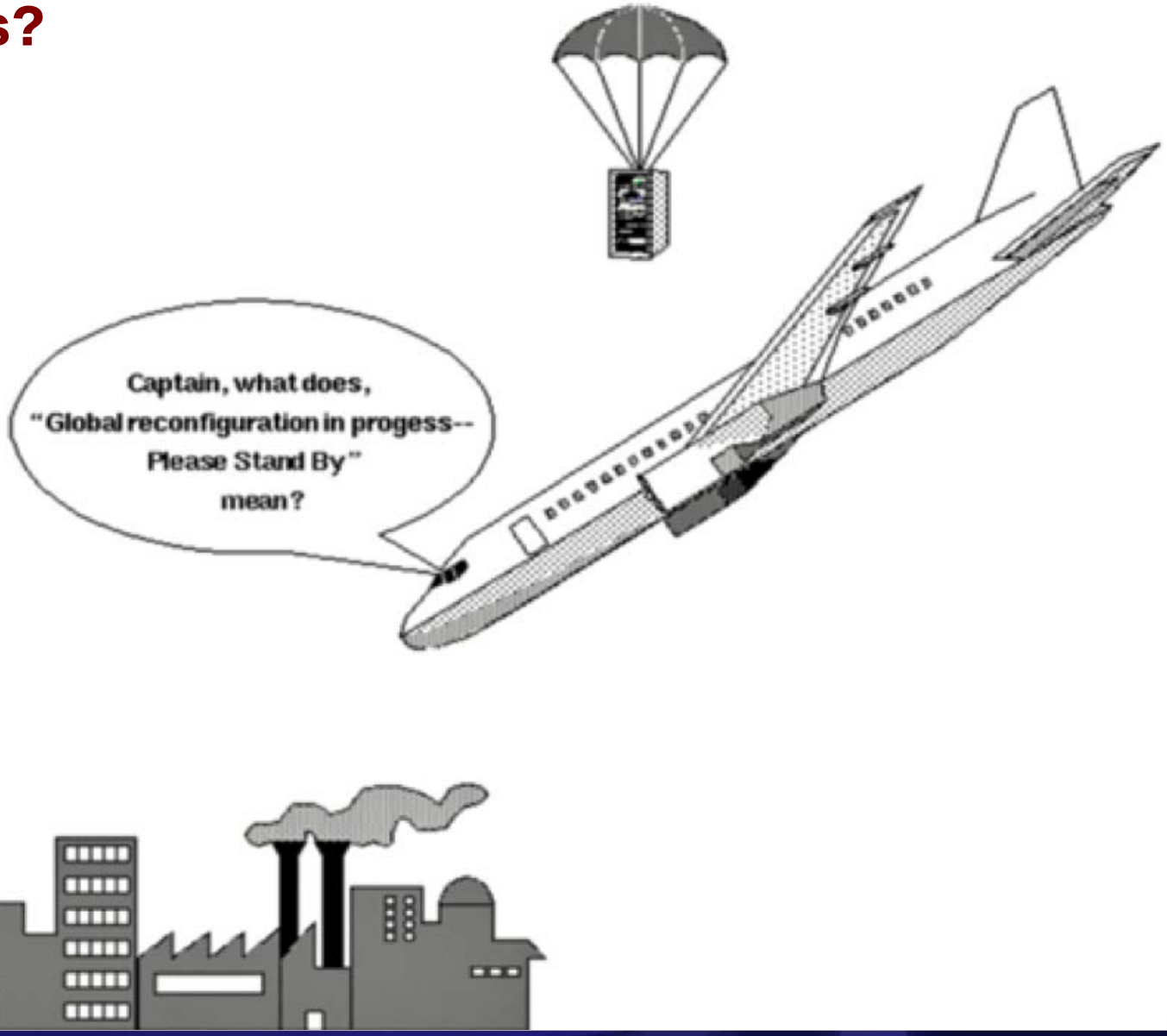
Identified by NAVSEA / NOSSA

Life Cycle Phase	Mitigation
Acquisition	Avoid software of unknown pedigree/provenance
Architecture	Perform architectural hazard/risk analysis and modeling
	Implement execution environment constraints
	Block access to unused functions
Assembly	Use COTS SDKs to enhance components
	Implement checksum monitors/code signatures on components, data to indicate tampering/corruption
	Add explicit, specific error and exception handling logic

Recap

- Safety-critical software is more vulnerable and exposed than ever
  - Commodity components and technologies
  - Networking and remote accessibility
  - Increasing dependence on pure software control without manual/mechanical backup
- Traditional security is inadequate for safety-critical software
- Both safety-critical and security-critical software need to *survive* incidents, whether caused by hazards or threats
- ***Survivability*** is “the new black”

Questions?


Captain, what does,
"Global reconfiguration in progess--
Please Stand By"
mean?