Secure, Adaptable, Resilient, and Capable Systems (SARC)[1]
Vision & Priority

Brad Martin
Program Manager
DARPA I2O

[1]SARC Analytic Framework, Scherlis et al., DARPA, September 2022

Secure, Adaptable, Resilient, and Capable Systems (SARC)
Technical Goals and Challenges

SARC **attributes**	**Goals**	**Challenges**
Secure	• Higher levels of security • Rapid T&E judgments • Confident T&E judgments	• Attack surfaces expanding, not shrinking • Extensive internal attack surfaces • Software, hardware, apertures
Adaptable	• Rapid and continuous adaptation to nimble threats • Safe and rapid pivots in mission conop • Quick inclusion of new technical capabilities	• Technical architecture requires explicit planning • Loss of knowledge • Measurement and risk assessment
Resilient	• Operation through attacks, despite compromise • Higher levels of interconnection	• Resiliency retrofit mostly infeasible • Technical architecture is a key determiner
Capable	• Greater scale, affordably • Rapid-tempo engineering	• Tools and abstractions for increased complexity

Secure, Adaptable, Resilient, and Capable Systems (SARC)

Key Technical Ideas

Seven key <u>technical ideas</u> in support of SARC goals

1. Direct technical evidence
2. Combined software/hardware security
3. Assays for technical architecture
4. Ultra-granular system configuration management
5. Composable and integrated engineering models
6. Engineering tool chains for rapid iteration
7. Progress measures for iterative practice

The four SARC elements build on similar <u>foundations</u> in technology and practice

What we build:

- **Technical architecture**
- **Software abstractions**
- **Hardware, firmware, systems software**

How we build it:

- **Iterative incremental practice and tools**

How it is sourced:

- **Software supply chain practices**

How we assess it:

- **Direct engineering evidence**

The SARC analytic framework development is based on extensive engagement with diverse technical stakeholders in DARPA, DoD, partners, tech firms, etc.

Secure, Adaptable, Resilient, and Capable Systems (SARC)
Priority Areas for Technical Research

How we build it: Iterative incremental practices and tools

Safe and powerful languages
- With performance and productivity benefit

Integrated range tests with analysis and *in vitro*
- Improve validity and tempo, enabling continuous T&E

Legacy code – recover from information loss
- Recover the legacy into evolvable systems with continuous T&E

SARC power tools
- Open/shared frameworks for evidence mgmt.

What we build: Technical architecture

Architecture patterns for trusted enclaves
- Building blocks for safe fusing of data/analytic enclaves

Secure ultra-granular highly distributed systems
- Resilient designs, including distributed autonomy

Architectural modeling and analysis tools
- Predictive models for SARC characteristics

What we build: Hardware, firmware, systems software

Hardware isolation and protection
- Close the expanding attack surface for optimized hardware

Interconnection improvements
- Develop crossbar models that avoid exposures of buses

Runtime protections
- Monitoring, tagging, static/dynamic root-of-trust

How it is sourced: Software supply chain practices

Securing open source
- Evidence-based security for open source, integrated and side-car

Securing granular code elements
- Ultra-granular provenance for "invisible" software bill of materials

Protected evidence
- Mathematically-based security claims for protected intellectual property (IP)

How we assess it: Direct engineering evidence

Cyber risk assessment, revisited
- Analytic assays for both external and internal attack surfaces

The evidence-based software deliverable
- Formal/informal approaches to "engineering data" in acquisition

Integrated formal methods
- Scale and integrate formal methods (FM) into baseline tooling and practice

What we build: Software abstractions

Abstractions tailored to mission capabilities
- AI engineering, human-system, enclaves, crossbar, distributed, etc.

Engineering through multiple models
- Open framework for DoD model-based engineering capabilities

System performance enhancers
- Algorithmic, AI, and language tools for super-performance

Secure, Adaptable, Resilient, and Capable Systems (SARC)

Informed by DARPA Program Experience

SARC technical elements	**DARPA programs** (I2O, MTO)	Security	Adaptability	Resiliency	Capability
What we build: • **Technical architecture** • **Software abstractions**	**CASE** (essential architecture role in resiliency) **ANSR** (new software abstractions for trustworthy AI) **CRASH** (feasibility and benefits of metadata tags for security) **HARDEN** (strong software abstractions) **SafeDocs** (securing data exchange connections in systems)	• Architecture patterns that predict security • Protect abstractions to reduce vulnerability	• Architecture patterns that enhance adaptability	• Architecture patterns that enable and enhance resiliency	• Abstractions enable increased capability, including for AI reliant systems
What we build: • **Hardware, firmware, systems software**	**GAPS** (hardware architecture to enhance data separation) **SSITH** (hardware secured from common software vulnerabilities) **AISS** (automation to enhance security in hardware designs) **FRANC** (new hardware architectural concepts)	• Enable hardware engineering choices that can enhance security			• Facilitate task-focused hardware designs
How we build it: • **Iterative incremental practice and tools**	**ARCOS** (rethinking process to support evidence capture) **SDH** (rapid and effective design cycles for hardware)		• Enable rapid design cycles for hardware		• Enhance capability by improving
Where it comes from: • **Software supply chain practices**	**ConSec** (configurations of diverse components) **V-SPELLS** (reversing component structures in legacy software) **SocialCyber AIE** (situation awareness for open source supply chains) **SymCPS** (cyber-physical configuration engineering)	• Better identify and mitigate security issues deep in software supply chains	• Reverse engineering to facilitates software adaptability		
How we assess it: • **Direct engineering evidence**	**AA** (evidence and direct analysis for trustworthy machine learning) **ARCOS** (integrating structures for evidence) **HACMS** (security through formal evidence) **SIEVE** (cryptographic techniques for IP-secure attestation) **CHESS** (tool-enhanced vulnerability discovery) **AMP** (binary program understanding and repair for existing software)	• Create evidence to support direct judgments for security attributes	• Use technical evidence to speed adaptability through rapid T&E cycles		• Enhance capability for AI reliant systems

High-Assurance Cyber Military Systems

Verified high assurance at system level (HACMS)
From formal methods + verified OS + architecture design

- Highly assured seL4 OS (using formal methods)
  - Enforces architectural separation of critical components
- Withstood hacker attacks at 2021 DefCon
  - "*For the first time ever, DARPA let all comers try to hack its HACMS high assurance software.*" —AIR FORCE Magazine


AIR FORCE MAGAZINE
Login Search
NEWS DAILY REPORT MAGAZINE DOCUMENTS ABOUT US AFGHANISTAN

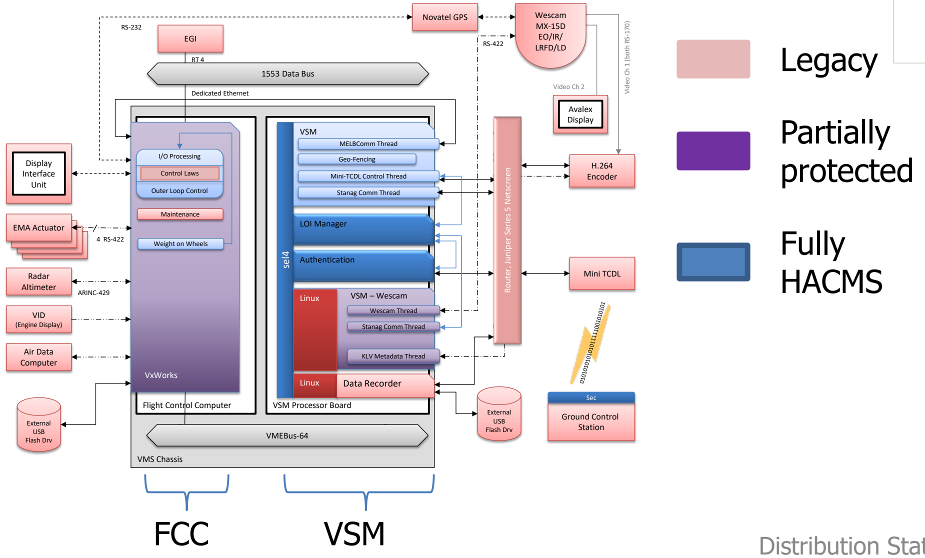
For the first time ever, DARPA let all-comers try to hack its HACMS high assurance software — seen here controlling a flight disabed drone. Photo courtesy DARPA via Twitter

DARPA Drone Cybersecurity Software Foils Hackers in Demo

SHARE ARTICLE

Aug. 13, 2021 | By Shaun Waterman

Phase 3 Architecture



- Attacker uses memory protection vulnerability to gain control of legacy application, and attempts to break out of the Linux virtual machine (VM) and access encryption keys

- **Without HACMS:** *attacker can overwrite key* and take control
- **With HACMS:** legacy application compromised, but *attacker cannot access key – and no other components affected*

DARPA

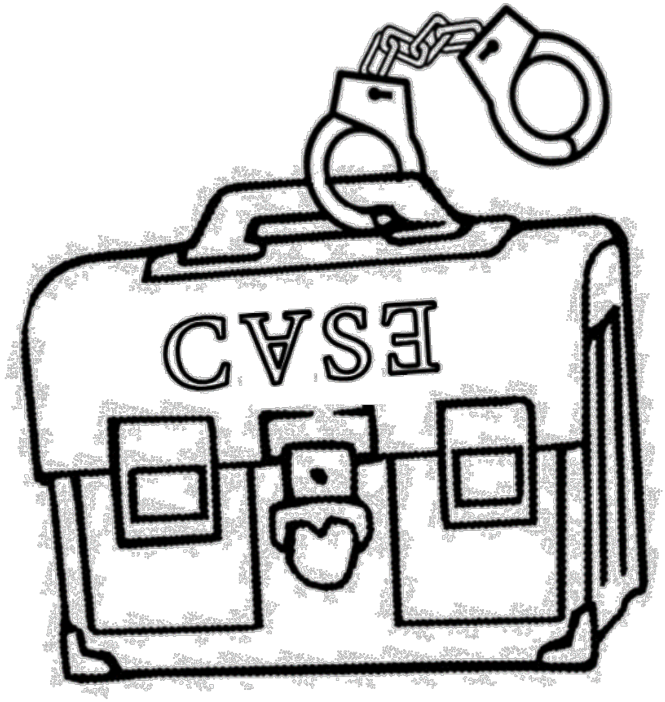
Cyber Assured Systems Engineering

Develop model-based systems engineering tools and workflow to make the HACMS approach repeatable, scalable, more incremental

End-to-End Integrated Formal Verification

- system properties (AGREE)
- architecture properties (AGREE)
- high-assurance components | legacy components
- HAMR correspondence proof
- CAmkES translation proof
- seL4 initializer proof
- seL4 proof

assurance case



sel4 ® - DARPA I2O Related Initiatives

Extending Assurance of kernel Software to the Instruction Level (EASIL)

- Performer: Siege Technologies
- Research: I2O Seedling
- Summary:
  - Connect the formal proof and properties at the kernel binary level (i.e., seL4) with those at the instruction set architecture (ISA) level
  - Extend formally verified software assurance to the ISA (or even micro architecture) level
  - Identify reusable and automatable components in the solution to enable automated targeting of assurance to another ISA or ISA model
  - Investigate and recommend hardware and software packages (as a 'product') for DoD developers

A Secure Distributed Computing Middleware for the seL4 Ecosystem

- Performer: Real-Time Innovations (RTI)
- Research: DARPA SBIR
- Summary:
  - Developed and contributed open-source, secure version of their commercial DO178-C Level A certifiable *RTI Connext DDS Micro*
  - Provided set of supporting tools to the seL4 developer community to help foster adoption of this new technology and the underlying seL4 kernel

Establishment of a US/DoD-based seL4® Trusted Computing Center of Excellence (TCCoE)

- Performer: Intelligent Automation Inc., dba BlueHalo
- Research: DARPA SBIR
- Summary:
  - Training materials and annual Summit with participants highlighting latest research
  - Establishment of software repository giving developers sel4 code exemplars to speed their understanding of the technology

sel4 ® - DARPA I2O Related Initiatives

AutoMatEd THeorY SubsTiution (AMETHYST)

- Performer: Two-Six Labs
- Research: DARPA AIE – PEARLS
- Summary:
  - Researching the feasibility of proof repair for C loop invariants in the seL4 microkernel, via the Isabelle/HOL proof system

Exploring network stack for seL4

- Performer: Galois
- Research: CASE ECP
- Summary:
  - Investigating reuse or writing of drivers to integrate with sel4 build system
  - Implementing the selected socket API in Rust in order to seamlessly integrate with Rust applications and investigate the usability of libsel4osapi which provides a non-posix socket interface in C in order to implement the selected socket API in Rust with the appropriate seL4 bindings

Exploring binary-analysis techniques with bottom-up formal verification

- Performer: Georgia Tech, UT Dallas, TrustedST
- Research: V-SPELLS, PEARLS AIE (TrustedST)
- Summary:
  - sel4 used as 'canonical' example in developing integrated pipeline which leverage source- and binary-based analysis and associated proof tools

www.darpa.mil