Compartmentalization and Privilege Management (CPM)

Howard Shrobe
DARPA
Information Innovation Office (I2O)

Proposers Day

March 2023

CPM program objective

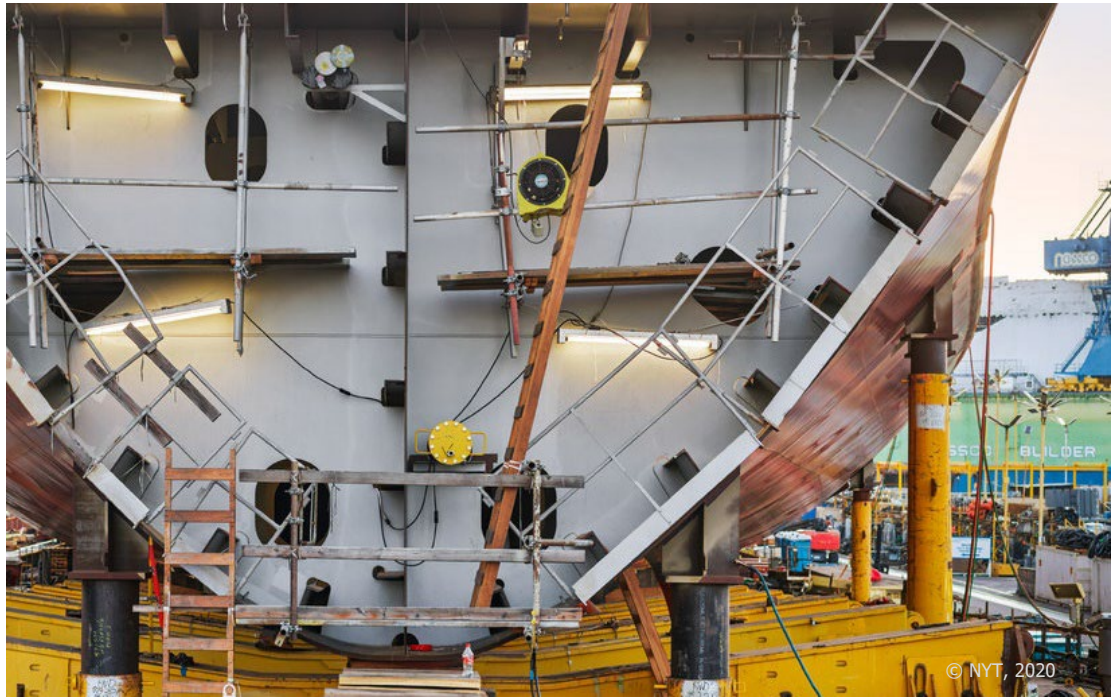
Develop a set of analysis tools, along with supporting hardware and software infrastructure, to *automatically* compartmentalize large legacy software systems into *performant*, *limited-privilege*, *fine-grained* compartments that **prevent initial penetrations from turning into successful cyber attacks**

A compartment analogy

How ships used to be built:

A single shot at the water line could sink the ship

Modern ships are divided into many compartments, each separated by a watertight bulkhead:

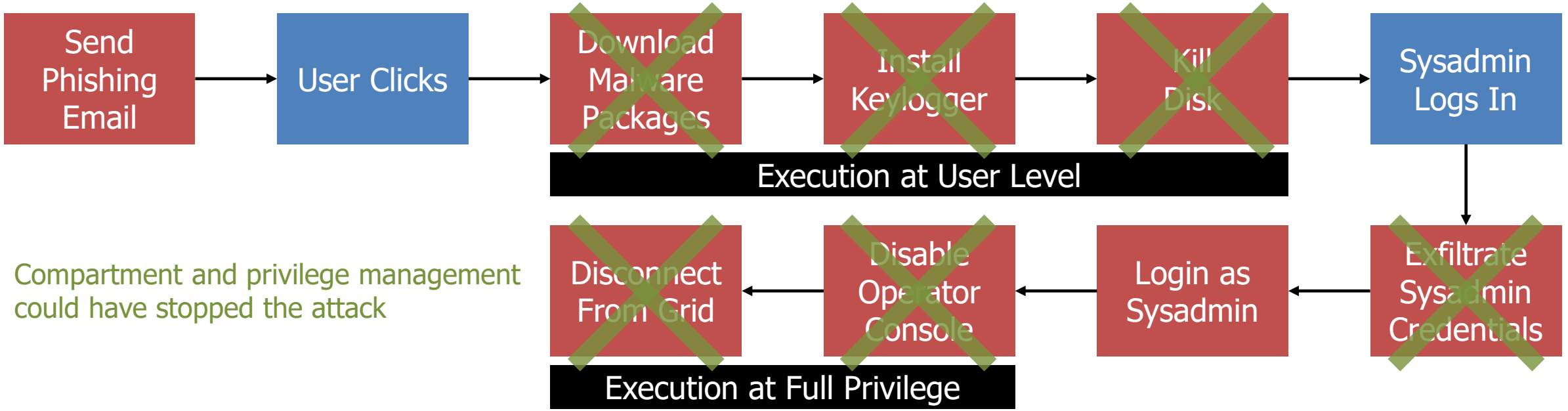


No single failure can sink the ship

Example: Cyber attack against Ukraine power grid

Lack of compartment and privilege management led to successful cyber attack

- December 2015, Russian cyber hackers turned off power to Kyiv for several hours*
- Cyber attack was months-long campaign to gain control of the software and hardware that administers the Kyiv power grid

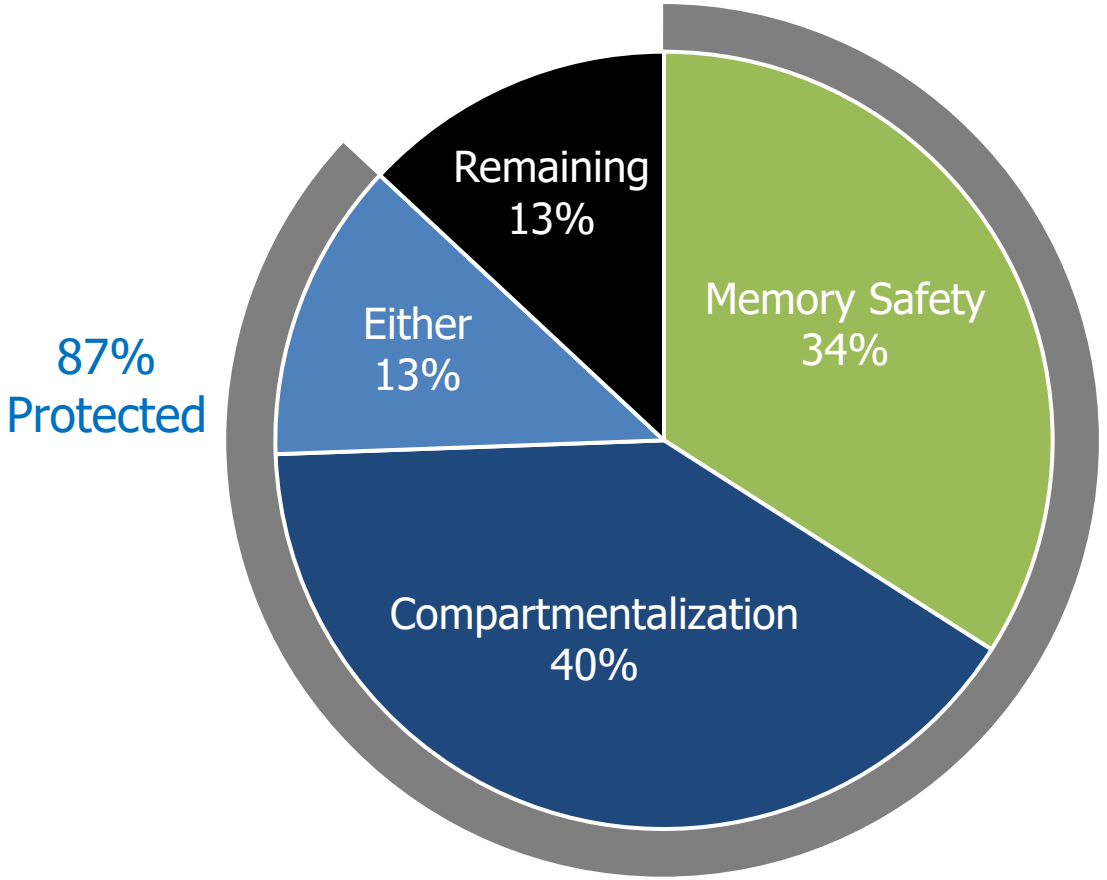


Compartment and privilege management could have stopped the attack

* *Inside the Cunning, Unprecedented Hack of Ukraine's Power Grid*, Kim Zetter, Wired. 3 March 2016

Memory safety and compartmentalization would stop most campaigns

Mitigation analysis of Linux kernel high severity CVEs



87%
Protected

- Either: if memory safety or compartmentalization were present, the bug would not be exploitable
- Remaining: memory safety or compartmentalization would not mitigate the CVE

CVE: Common Vulnerabilities and Exposures

McKee, Derrick, et al. "Preventing kernel hacks with HAKC." Proceedings 2022 Network and Distributed System Security Symposium. NDSS. Vol. 22. 2022.

Relevance to DoD

Compartmentalization and privilege management **should** be the norm in system design*

- But it rarely is, particularly in **legacy** systems

Legacy systems over their lifetimes tend to become more unstructured and consequently less compartmentalized

- "In the long run every program becomes rococo - then **rubble**."
  Alan Perlis, former professor at Carnegie Mellon University and Yale, "Epigrams in Programming"

Software-only solutions are not performant; hardware support will be essential

Unusually long-lived critical legacy systems are the norm in DoD

Credit: Airman 1st Class Victor J Caputo/ commons.wikimedia.org.

B-52 designed in 1947, first flight in 1952, in service with USAF 1955, 76 still in inventory

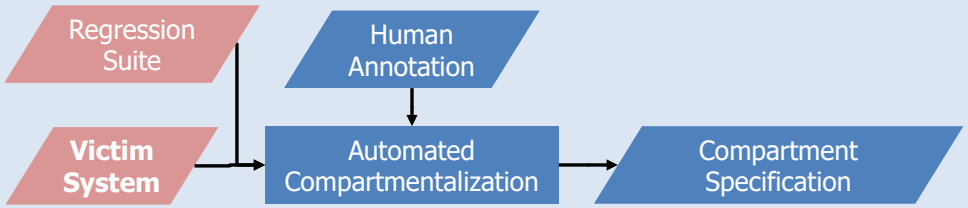
The scope of this problem is **unique to DoD** and will not be addressed by industry

* "The Protection of Information in Computer Systems", Saltzer & Schroeder Fourth ACM Symposium on Operating System Principles (October 1973). Revised version in *Communications of the ACM* 17, 7 (July 1974).

CPM program structure

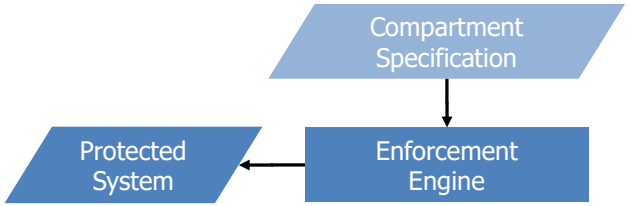
TA1: Automated Compartmentalization

- Develop software analysis tools that **automatically compartmentalize legacy code** and manage privilege levels to prevent initial penetrations from propagating



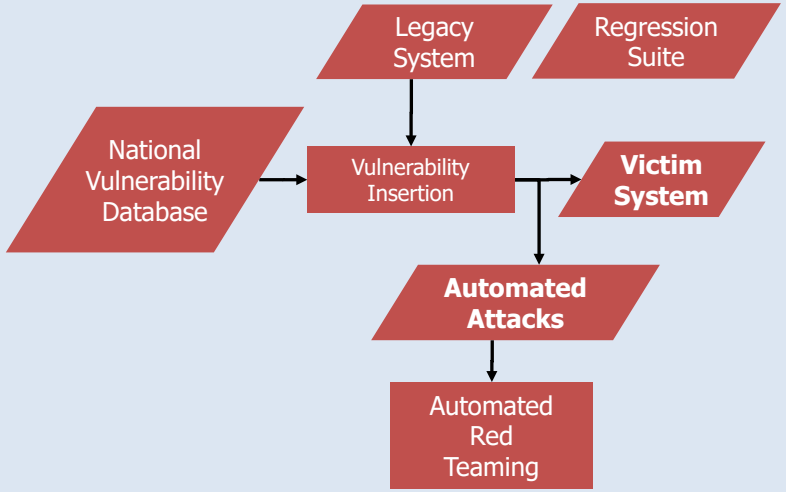
TA2: Enforcement

- Develop processor architectures and system software that will, with low overhead, **enforce** a compartment and privilege level regime that prevents initial penetrations from propagating into a successful cyber attack



TA3: Evaluation Support

- Develop a library of attack campaigns that can be used as open-source test cases for the combined TA1-TA2 systems. Evaluate the success rate and overhead of the systems defended by TA1—TA2 systems versus the baseline unprotected systems



TA4: DoD System Experimentation (delayed solicitation)

- Apply CPM tools to a modest number of existing, proprietary DoD systems and evaluate the difficulty and efficacy of defending these systems

TA: Technical Area

The CPM abstract model*: Compartments, principals, access rules

Principal
An active entity playing a specific role

Compartment
Collection of data with a specific purpose

Load	C-1	C-2	C-3	C-4	C-5	…	C-n
Principal-1	Y	Y	trap	trap	trap		trap
Principal-2	trap	Y	Y	Y	trap		trap
Principal-3	Y	trap	trap	trap	Y		trap
⋮							
Principal-n	trap	trap	Y	trap	trap		Y

- Everything, including data, code, Principals and Compartments, are represented as objects within a compartment
- Each executing thread has a Principal (who it's acting for and in what role)

Access rules

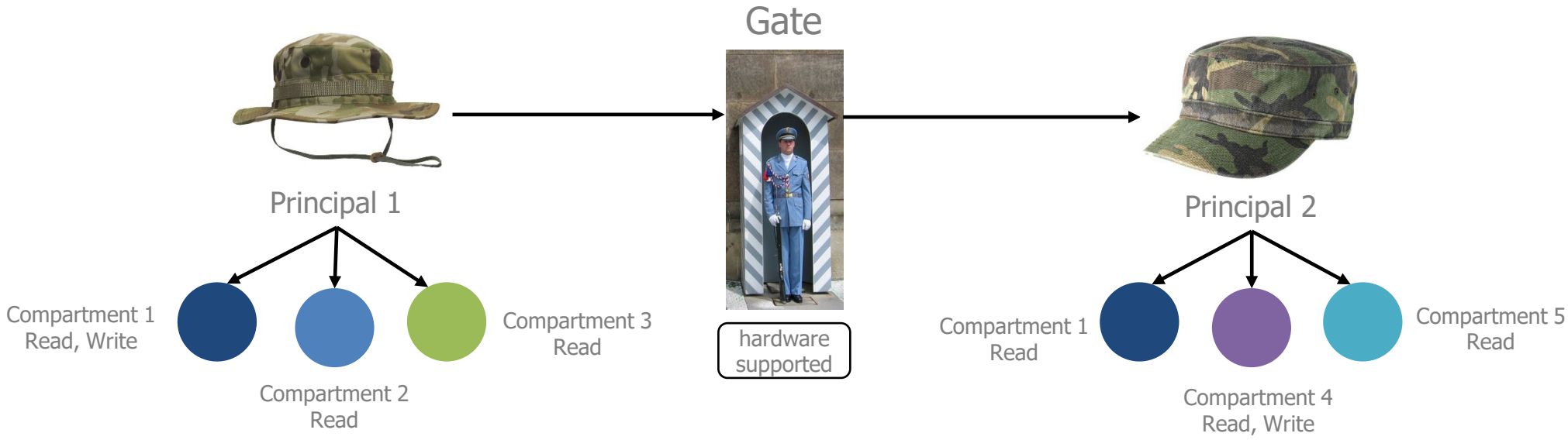
- For each compartment and each principal, a matrix specifying which operations are legitimate
- Collectively forms a policy restricting flow of data between compartments and integrity of data within each compartment
- Principal changes are controlled with "gate calls"

trap — A trap is a hardware generated interrupt of the running process, indicating that something has gone wrong. Control is diverted to a software "trap handler" that can either fix what's wrong or terminate the program.

* *Trust-management, intrusion-tolerance, accountability, and reconstitution architecture (TIARA),* Shrobe, Howard, Andre DeHon, and Thomas Knight. MASSACHUSETTS INST OF TECH CAMBRIDGE, 2009.

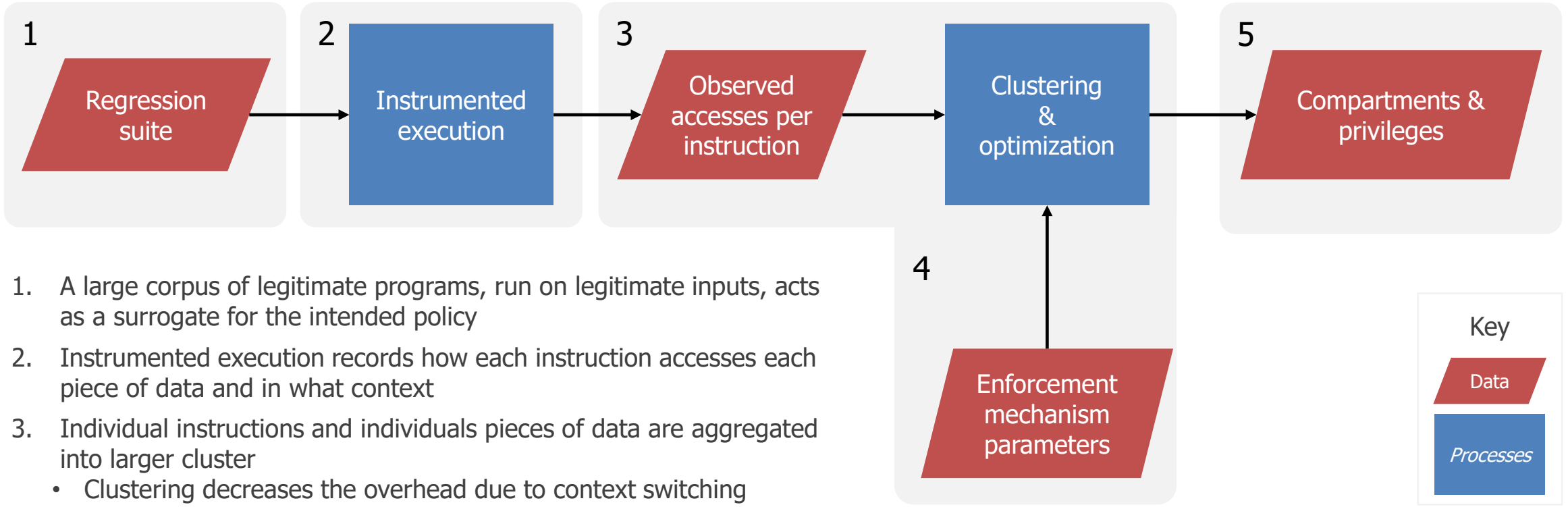
The CPM abstract model: Change of principal

Transition to a new principal can only be achieved through a "**gate call**" –
a procedure call that changes the principal for the duration of the called procedure



- The privileges accorded to the new principal are only those needed to perform the function of the procedure and are not strictly greater than those of the old principal
- Gate calls are themselves objects within a compartment and are subject to access control rules, so only some transitions are possible within each context

TA1: Automated compartmentalization – dynamic analysis approach



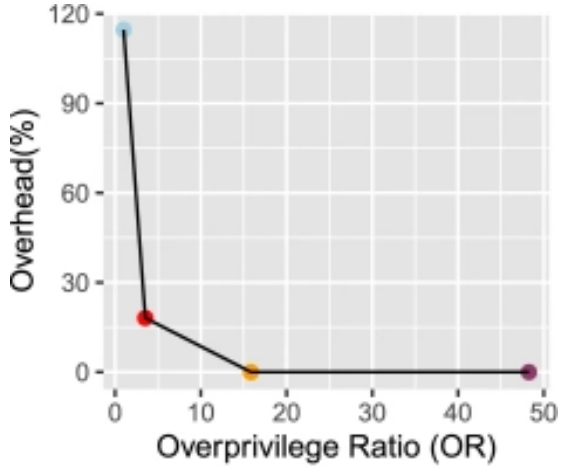
1. A large corpus of legitimate programs, run on legitimate inputs, acts as a surrogate for the intended policy
2. Instrumented execution records how each instruction accesses each piece of data and in what context
3. Individual instructions and individuals pieces of data are aggregated into larger cluster
   - Clustering decreases the overhead due to context switching
   - Clustering increases the "over-privileging" of each context
4. Optimization finds the best trade-off given the costs of the enforcement mechanism
5. Output of optimization is specification of compartments and enforceable policy for privileges

SCALPEL: Exploring the Limits of Tag-enforced Compartmentalization. Nick Roessler and André DeHon, University of Pennsylvania, USA

Examples of experimental compartmentalization systems

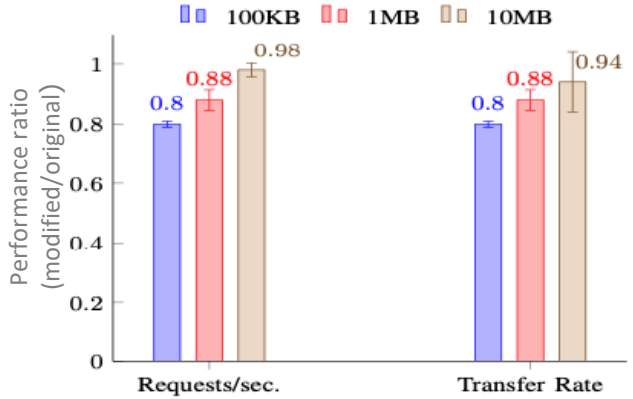
SCALPEL: Exploring the Limits of Tag-enforced Compartmentalization

Nick Roessler and André DeHon, University of Pennsylvania, USA



Preventing Kernel Hacks with HAKC

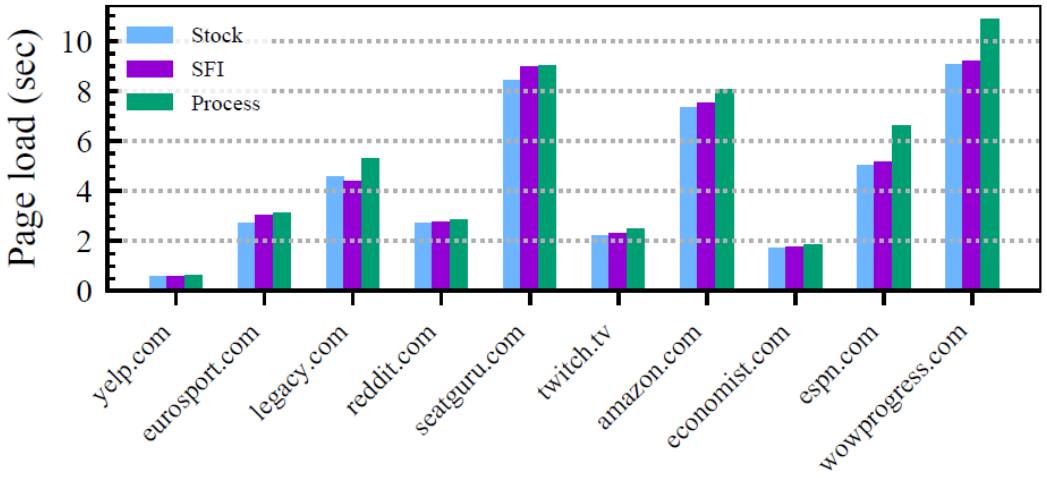
Derrick McKee*, Yianni Giannaris†, Carolina Ortega Perez†, Howard Shrobe†, Mathias Payer‡, Hamed Okhravi§, and Nathan Burow§
*Purdue University, †MIT CSAIL, ‡EPFL, §MIT Lincoln Laboratory



Overhead impacts normalized to unmodified kernel when transferring various sized payloads.

RLBOX: Retrofitting Fine Grain Isolation in the Firefox Renderer

Shavan Narayan and Craig Disselkoen, UC San Diego; Tal Garfinkel, Stanford University; Nathan Froyd and Eric Rahm, Mozilla; Sorin Lerner, UC San Diego; Hovav Shacham, UT Austin; Deian Stefan, UC San Diego



SFI: Software-based Fault Isolation (i.e., Native Client [NaCl] sandboxes)
Process: Isolation in individual processes with IPC communication and synchronization

TA1 challenges

Dynamic analysis may under-privilege

- The regression suite may never make an access that is actually legitimate

Static analysis may over-privilege

- It is computationally too expensive to do a completely accurate static analysis
- Therefore all analysis techniques approximate, allowing some accesses that aren't legitimate

CPM will develop techniques for merging the two, yielding a more accurate policy

Human annotation may be needed to communicate higher-level insights, but requiring too much may make the approach impractical

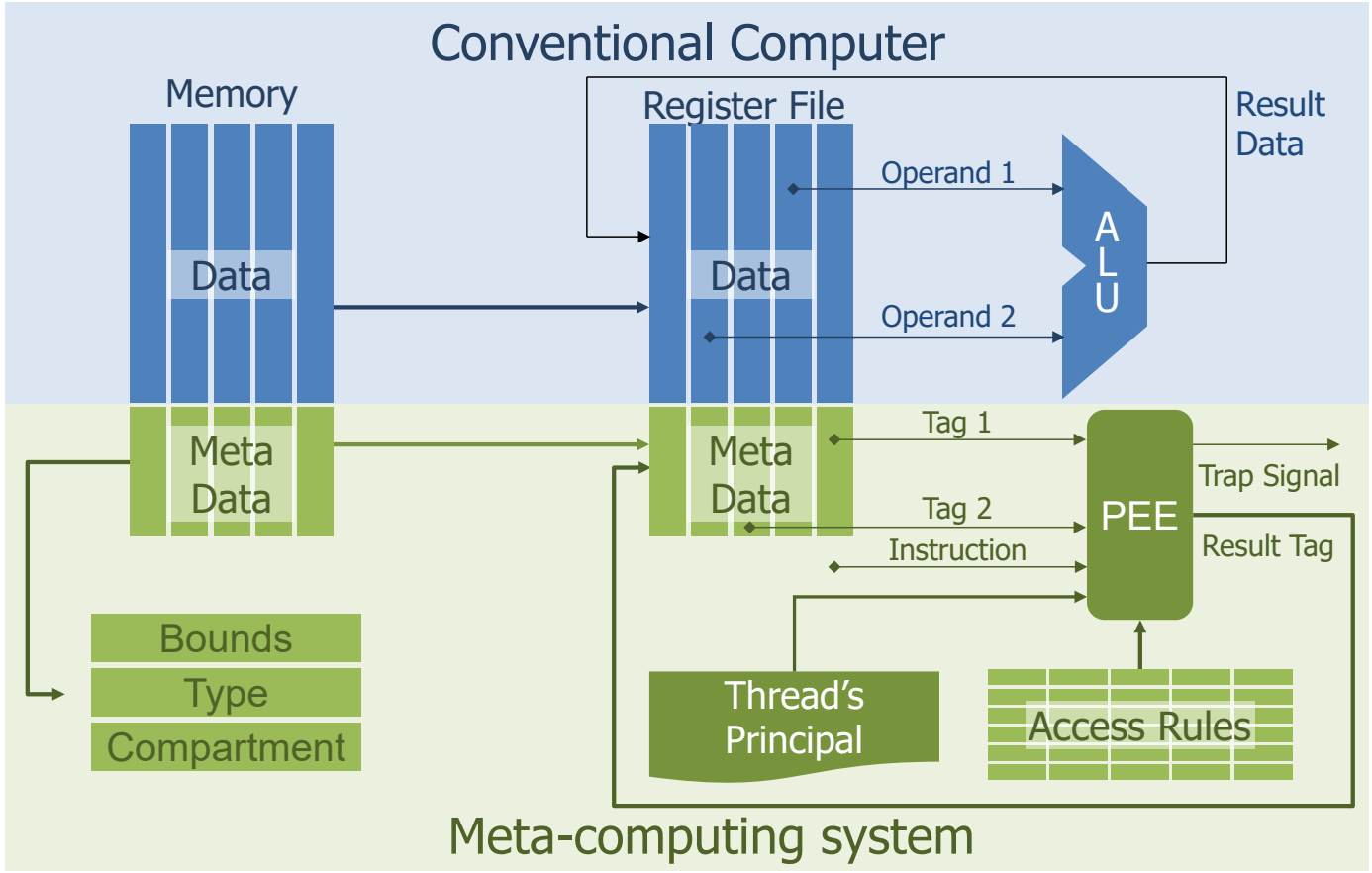
- For legacy systems the intended design is often obscured by years of patching and incremental development
- Nobody may actually understand the entire system design

Characteristics of a strong TA1 proposal

- Develop tools that can target multiple TA2 enforcement systems
- Present a clear path for scaling the proposed approach to very large code bases and to a variety of workloads (e.g., programs that are heavily interactive, Input/Output-intensive, pointer-intensive, etc.)
- Describe what sort of manual annotation is anticipated and how the need for manual annotation will be minimized
- Identify the key technical risks of the proposed approach, describe how they will be tracked, and suggest possible mitigations

TA2: Enforcement

- Compartment privileges and transition rules for principals must be enforced and require hardware assistance for performance*
  - Some support in existing hardware (e.g., Arm's pointer authentication and memory tagging instructions)
  - More robust and adaptable support evolving in research hardware such as Capabilities and Programmable Metadata Processing
- Available enforcement mechanisms expected to affect compartmentalization decisions



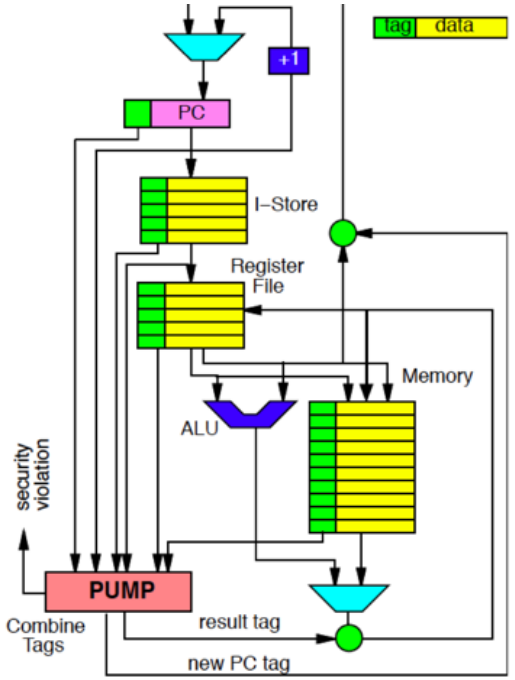
* "SoftBound: Highly compatible and complete spatial memory safety for C", Nagarakatte, Santosh, et al., *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2009

ALU: Arithmetic Logic Unit
PEE: Privilege Enforcement Engine

Examples of experimental enforcement systems

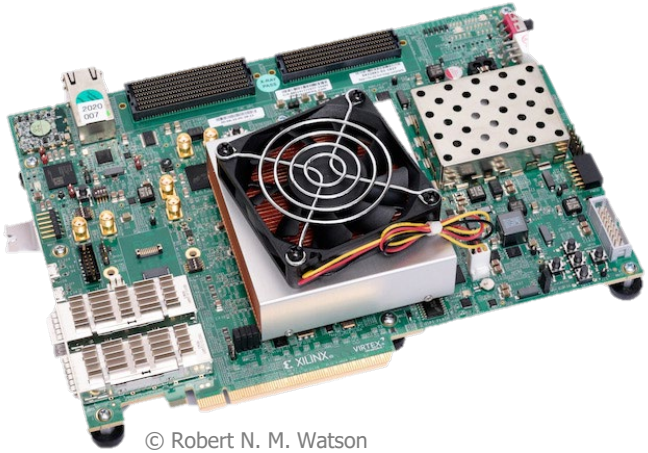
CRASH and SSITH systems (experimental)

Dover
A Metadata-Extended RISC-V



Enforces more complete model
- Less mature
- Programmable policies
- High memory overhead

CHERI RISC-V

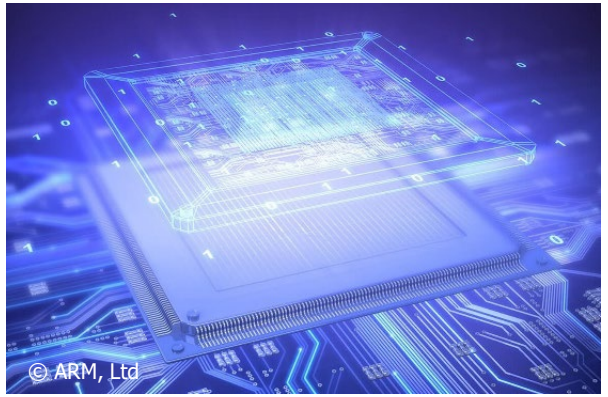


© Robert N. M. Watson

Enforces a more limited model
- Less context sensitive
- Open to "Confused Deputy" problems
- Temporal memory safety issues
- Performance still needs improvement

Commercial

HACKC
(simulated Arm 8.5A)



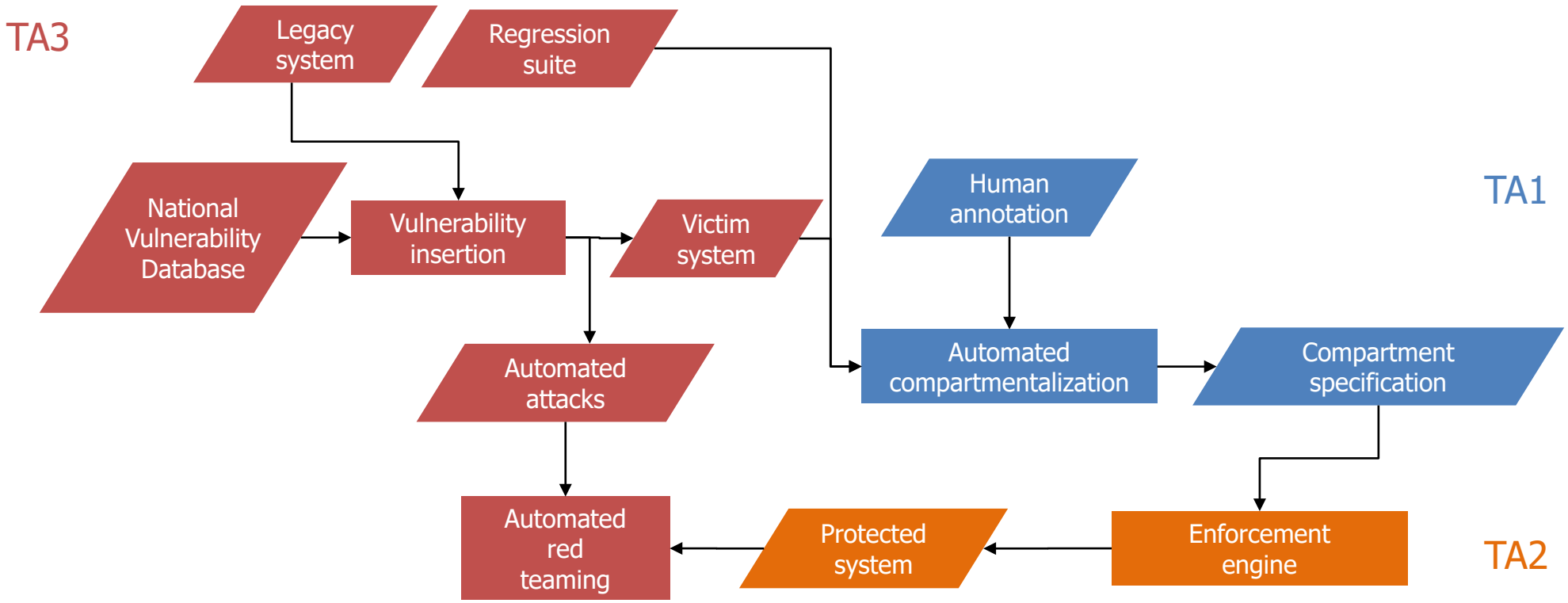
- Evaluated in two kernel modules
- Analysis done only in simulation

Clean-slate design of Resilient Adaptive, Secure Hosts	(CRASH)
System Security Integration Through Hardware and Firmware	(SSITH)
Capability Hardware Enhanced RISC Instructions	(CHERI)
Reduced Instruction Set Computer	(RISC)
Hardware-Assisted Kernel Compartmentalization	(HACKC)

Characteristics of a strong TA2 proposal

- Early simulated performance numbers for use by TA1 in determining optimal compartmentalization and privilege management strategies
- Plans for incorporating feedback from TA1 into the processor design and analysis process
- Timely hardware instantiations of the proposed processors, where timely means realistic, hardware-in-the-loop measurements of end-to-end systems can be conducted
- Descriptions of assumptions or limitations of the proposed approach to different workloads (e.g., frequent privilege policy changes, pointer-intensive, large vs small memory allocations, etc.)
- Identify the key technical risks of the proposed approach, describe how they will be tracked, and suggest possible mitigations

Experimental design



- Generate victim system and automate attacks
- Analyze protection
- Analyze performance

- Build the compartmentalization structure
- Build the protected system

Experimental design: Final outputs

Artifacts

Victim system

Automated attacks

Regression suite

Evaluation

CPM specification

Protected system

Automated compartmentalization

Enforcement

Metrics

Automated compartmentalization (TA1)

- % Lines of manual annotation
- Compartment size
- Over-privilege ratio
- Analysis time

Enforcement (TA2)

- % Attacks blocked
- CPU overhead
- Memory overhead

Characteristics of a strong TA3 proposal

- Describe how all of the program metrics will be measured, including those metrics that do not involve vulnerability testing
- Present a methodology for finding or creating vulnerable open-source systems (i.e., victim systems) and unclassified, non-sensitive attacks campaigns that are known to exploit victim system vulnerabilities. This includes user-space applications that are not currently selected as well as Linux
- Include an automated testing procedure for executing attack campaigns against victim systems and reporting outcomes
- Identify the key technical risks of the proposed approach, describe how they will be tracked, and suggest possible mitigations

A few words about technical risk

Reach far and risk failing

- If you aren't proposing things that might not work
- If your only risk is that another performer might not deliver

Then you are probably not going to wow us

Manage technical risk, don't avoid it

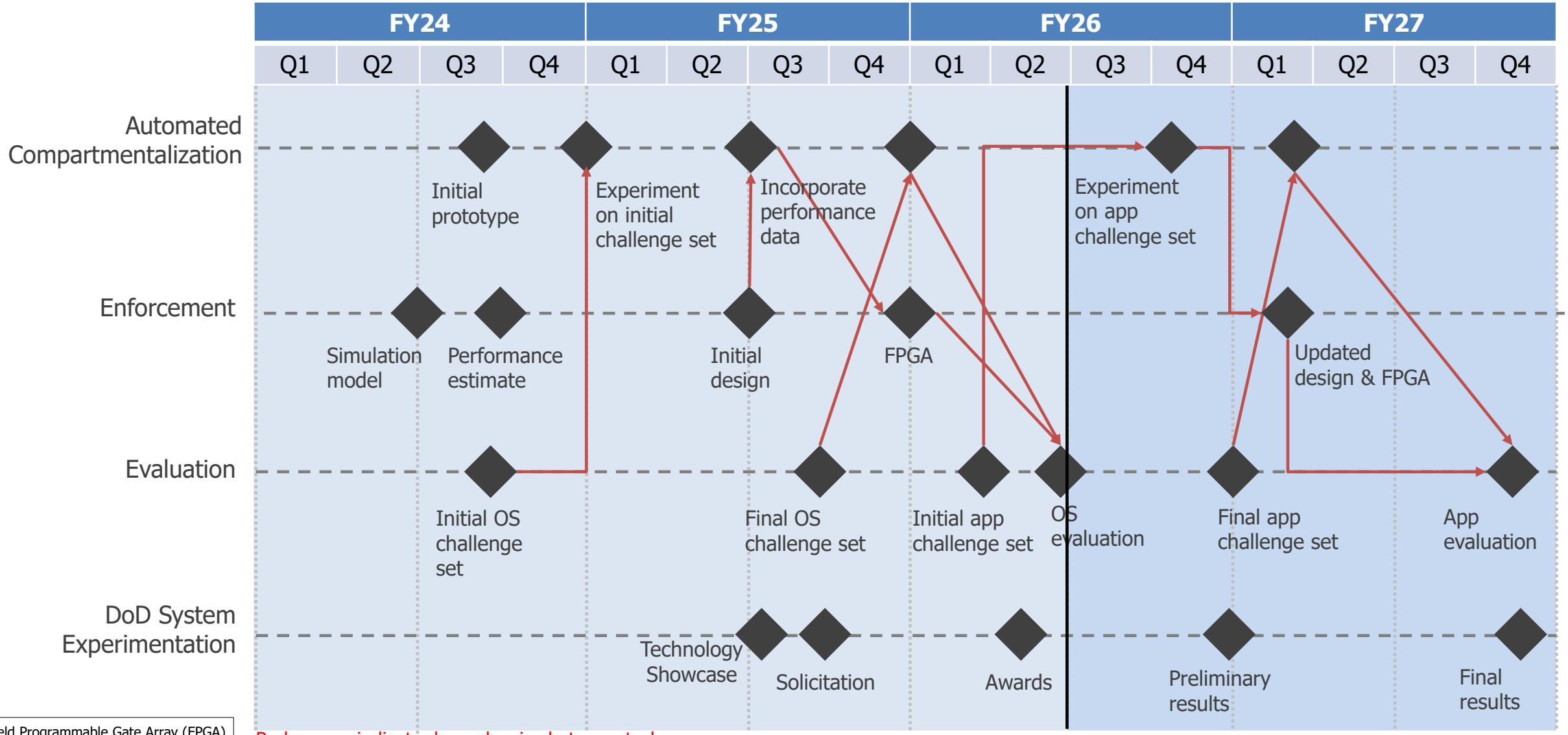
- Tell us what your risks are, particularly your technical risks
- Tell us how you will know if something isn't working out
- Tell us how you plan to respond when you realize something isn't working

Program metrics

Capability	Metric	Phase 1 -- OS	Phase 2 -- Applications
Performant	% attacks blocked	> 50%	> 85%
	CPU overhead*	< 15%	< 5%
	Memory overhead	< 50%	<25%
Fine-grained, limited-privilege	Compartment size	Individual function	Individual function
	Over privilege ratio	< 2%	< 1.5%
Automated compartmentalization	Analysis time	< 1 week	< 1 day
	% Lines of manual annotation	< 2%	< 0.2%

* This goal comes from the Microsoft BlueHat competitions

Milestones



Field Programmable Gate Array (FPGA)
Operating System (OS)

Red arrows indicate dependencies between tasks

Other relevant DARPA programs

DARPA program	Relevance to CPM program
Clean-slate design of Resilient Adaptive, Secure Hosts (CRASH)	Introduced idea of hardware enforced ”inherent security” through use of pervasive meta-data and policy engine
Mission-oriented Resilient Clouds (MRC)	Applied CRASH ideas in the distributed systems context
System Security Integration Through Hardware and Firmware (SSITH)	Developed more efficient meta-data oriented hardware enforcement with attention to SWAP (size weight and power).
High-Assurance Cyber Military Systems (HACMS)	Developed software enforced coarse grained compartmentalization
Verified Security and Performance Enhancement of Large Legacy Software (V-SPELLS)	Using decomposition of legacy software into separate modules to reduce verification burden

www.darpa.mil