Program Insights from CLOSURE / DARPA GAPS

Presentation at the DARPA V-SPELLS Kick-Off

July 28, 2021

Mr. Michael Kaplan
Scientific Research Analysis Sr. Manager

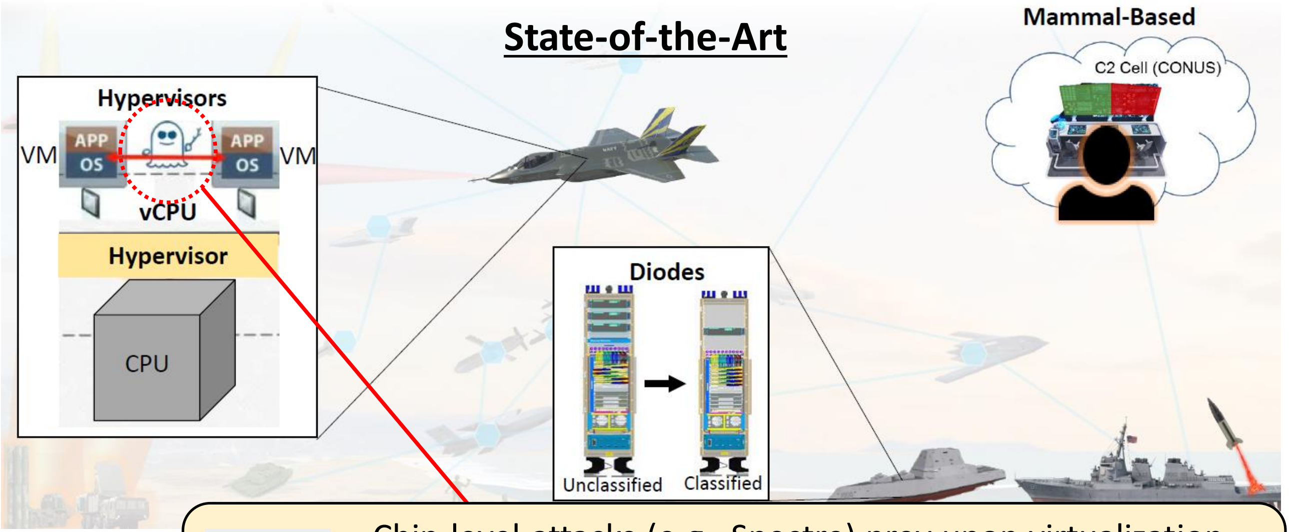
Dr. Rajesh Krishnan
Chief Research Scientist

Agenda

- Overview of DARPA GAPS Program and CLOSURE Project
- Technical Challenges and Solution Approach
- Demonstration
- Lessons Learned and Conclusions
- Q & A

Guaranteed Architectures for Physical Security (GAPS)

- **Problem:** Methodologies for ensuring data protections in DoD and commercial cross-domain (XD) systems are insufficient
  - Guarantees not traceable to source code/model, arduous accreditation effort
  - Expensive, complex deployment, inflexible to policy changes
- **GAPS Approach:** Novel co-design tools for verifiable partitioning of functionality with controlled data sharing across physically-isolated compartments



Source: GAPS

Chip-level attacks (e.g., Spectre) prey upon virtualization-based solutions. GAPS is predicated on strong hardware-based physical isolation immune to such attacks.

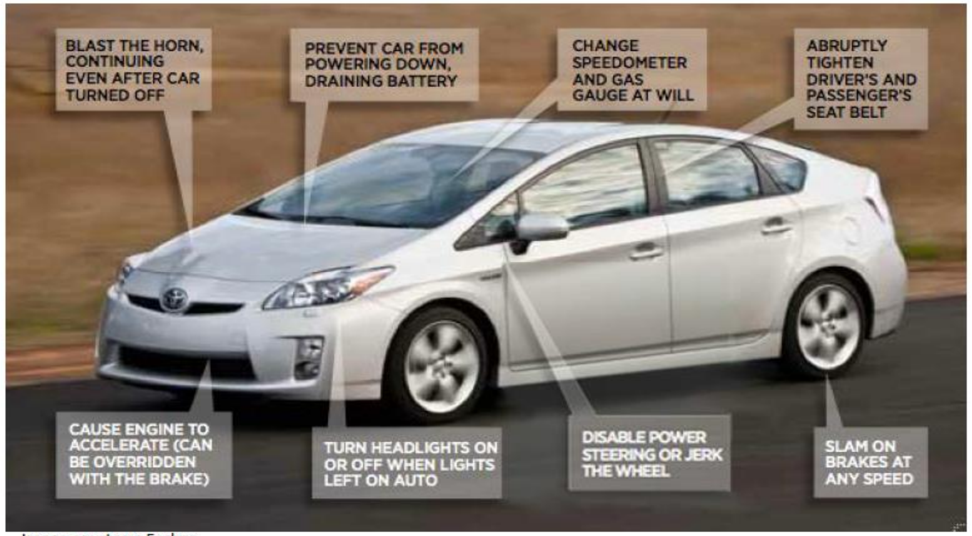


Image courtesy: Forbes

Source: GAPS Proposers' Day

Peraton LABS

Program Manager: Mr. Walter Weiss, DARPA MTO

GAPS Programmatics

Program Structure

Technical Areas	Current Performers	Solution
TA1: Components and Interfaces for strong isolation and high-speed interconnects	GE Research	**MIND**: Monitoring & Inspection Device (Ethernet/IP)
	Mercury Systems	**ILIP**: InLine Interface Processor (PCIe)
	Intel	**ESCAPE:** Extended Secure Capabilities Architecture Platform and Evaluation (UPI)
TA2: Co-Design Tools with novel language extensions for correct-by-construction compilation of cross-domain applications	Peraton Labs	**CLOSURE**: Cross-domain Language-extensions for Optimal SecUre Refactoring and Execution
TA3: Integration and Validation	Northrop Grumman	System Integration and Validation

Schedule

	Enclaves	Languages	Link Protocols	Bandwidth	Dates
Phase 1	2	1	1	100 Mbps	9/19-3/21
Phase 2	3	2	3	1 Gbps	3/21-9/22
Phase 3	4	2+	4	10 Gbps	9/22-3/24

Peraton LABS

Pain Points in Cross-Domain Systems Development

Cross-Domain Application Owner

- Inefficient lifecycle management
- Changing requirements impose re-engineering and accreditation cost

1. Requirements →

Systems / Security Analyst

- Lack of design / modeling tools to express cross-domain intent, evaluate partitioning feasibility, and do performance trade-offs

2. Design →

Hardware/Software Developers

- Lack of tools to express and handle data sharing concerns in program; labor-intensive program analyses, refactoring, hardware co-design

3. Cross-Domain System →

Test/Verification Authority

- Limited test and verification tools; poor traceability to source; catching incorrect rules, misconfiguration, and bugs involves tedious manual effort

4. Authority to Operate →

Admin/Operator

- Tedious manual configuration and testing; performance bottlenecks

5. System Operational →

CLOSURE is a software toolchain—a suite of program analysis, guided refactoring, partitioning, code generation, verification, and compilation tools—that addresses critical technology gaps affecting developers, users, and operators of cross-domain systems, which require guaranteed enforcement of data sharing policies via hardware means.

Technical Challenges and Architecture

CLOSURE covers a general problem in secure functional partitioning—with application to avionics platforms, mosaic warfare, coalition missions, critical infrastructure, healthcare, and other domains—where we need fine-grained controls on information sharing and rapid adaptation of software to changing requirements.

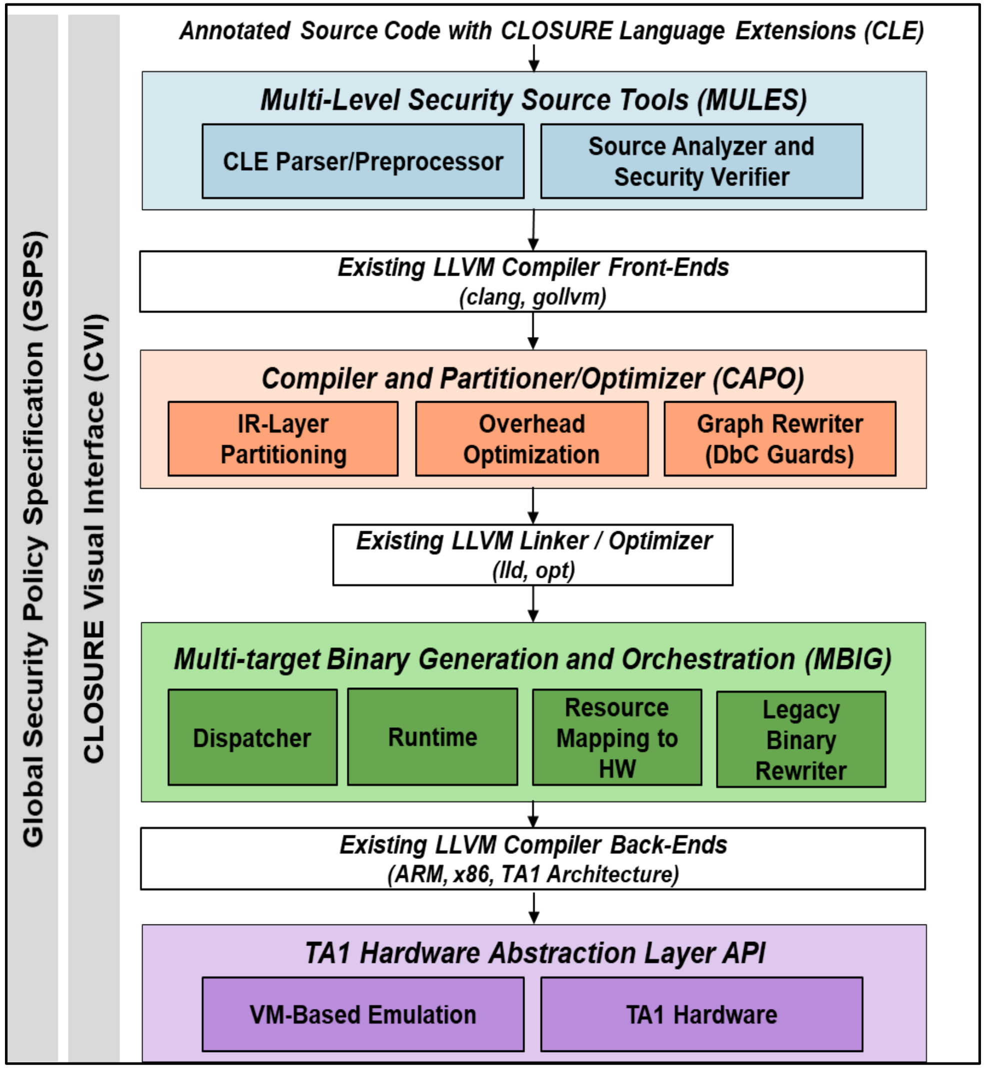
Expression: how to design intuitive language extensions to specify XD security intent

Automation: how to reduce developer effort via automated code generation, refactoring, and end-to-end optimization

Program analysis and guided refactoring: how to identify conflicts in annotated programs, provide actionable refactoring

Verification: how to argue formal correctness and compliance of partitioned programs

Ecosystem: how to foster adoption and participation from application, toolchain, and hardware developers



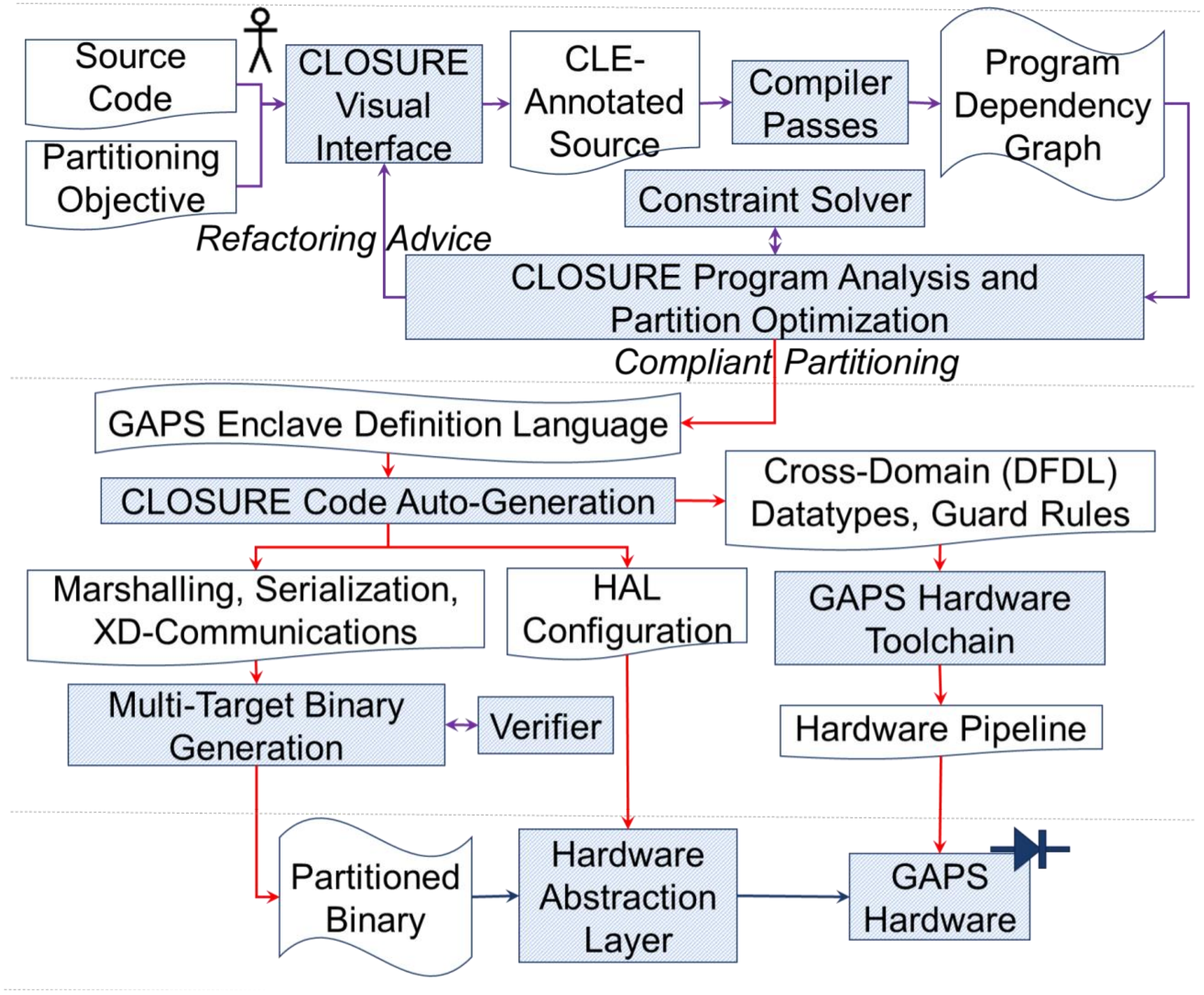
Modular system using established technologies

CLOSURE Workflow

Annotation-driven development for correct-by-construction partitions with interactive feedback for guided refactoring

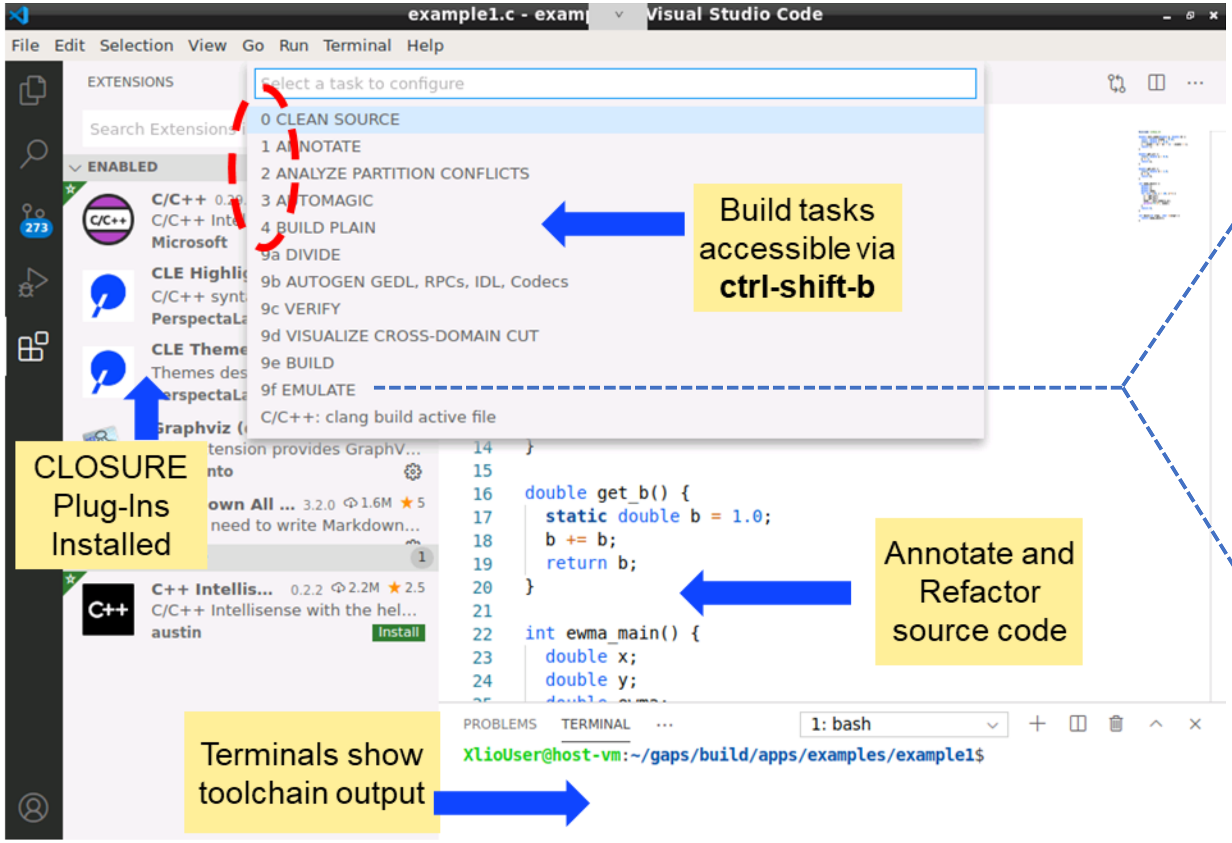
Automated generation of cross-domain artifacts, compilation, and verification of partitioned program

Seamless support for heterogeneous GAPS hardware architectures and emulation for pre-deployment testing

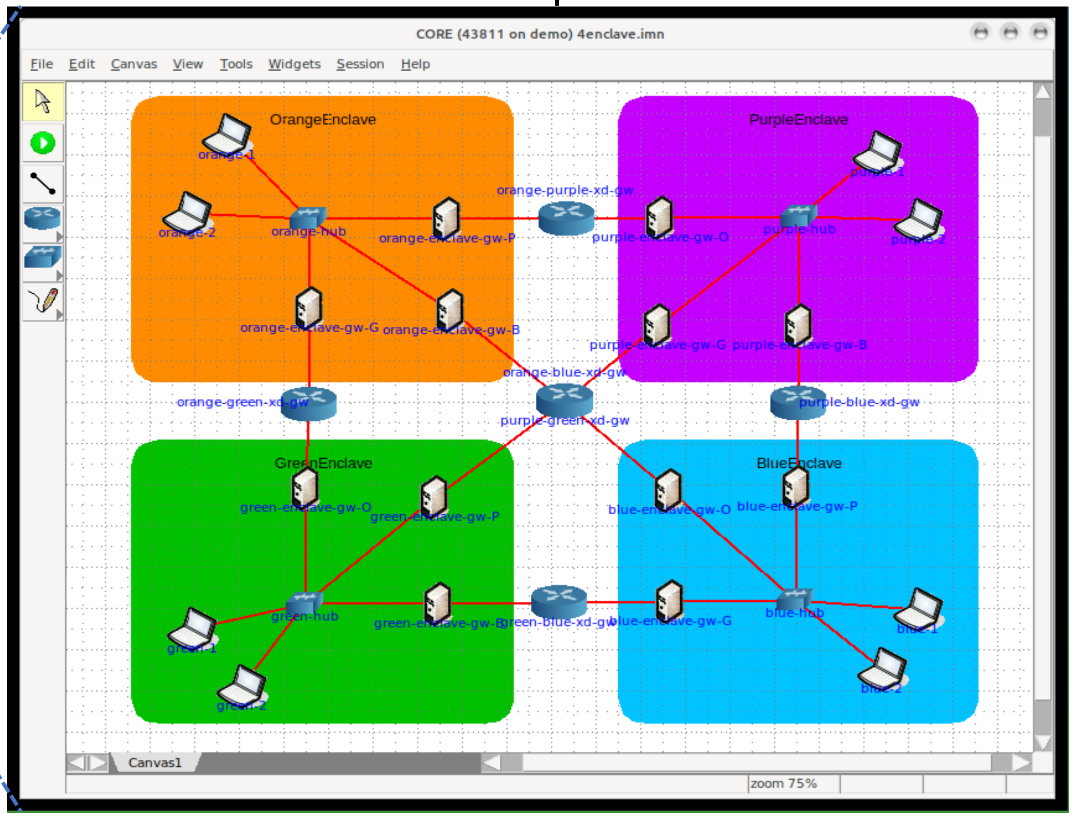


CLOSURE Visual Interface (CVI) and Emulator

CVI based on widely used VSCode platform



Emulator for multiple architectures



- Docker containers for easy CLOSURE toolchain installation
- Developer starts CVI and works on application source code
- CVI supports entire workflow from annotation to testing

- Comprehensive end-to-end testing prior to deployment
- Diverse GAPS hardware and different host architectures (QEMU)
- Scales to distributed multi-domain scenarios

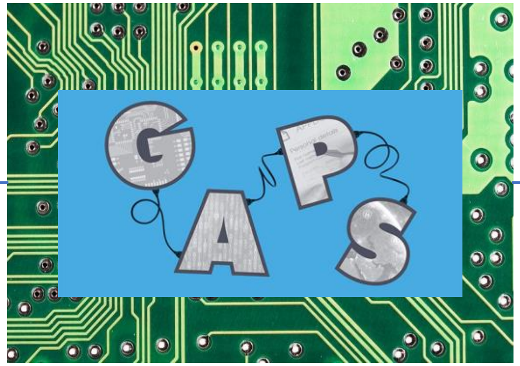
Peraton LABS

Annotation-Driven Program Partitioning

```
double get_a() {
#pragma cle begin ORANGE
  static double a = 0.0;
#pragma cle end ORANGE
  a += 1;
  return a;
}
double get_b() {
#pragma cle begin PURPLE
  static double b = 1.0;
#pragma cle end PURPLE
  b += b;
  return b;
}
int ewma_main() {
  double x;
  double y;
#pragma cle begin ORANGE
  double ewma;
#pragma cle end ORANGE
  for (int i=0; i < 10; i++
    x = get_a();
    y = get_b();
    ewma = calc_ewma(x,y);
    printf("%f\n", ewma);
  }
  return 0;
}
```

Developer annotates original source code to express cross-domain security intent

Enclave 1
(orange)



Enclave 2
(purple)

Automated program rewriting and code generation by CLOSURE tooling supports correct, concurrent execution of partitioned program binaries

CLOSURE Language Extensions (CLE)

- Enables the expression of cross-domain security concerns. Overlays existing code with industry standard compilers
  - Currently supports C, Java (in progress), and Message Flow Models
  - Reuse of CLE abstractions and concepts across programming language
- For C applications, MULES converts CLE to Clang attributes
  - No modification to LLVM/Clang
  - CLE labels flow down to IR for analysis
- Annotations enable toolchain to verify policy and identify conflicts prior to partitioning at function boundaries (for C)

variable annotations

Defines CLE labels and associated security policies

```
#pragma cle def ORANGE {
 "level":"orange",
 "cdf": [
   {"remotelevel":"purple",
    "direction": "egress",
    "guarddirective": { "operation": "allow"}}
 ] }
```

Annotate code with CLE label

```
#pragma cle begin ORANGE
double precise_readings[128];
#pragma cle end ORANGE
```

function annotations

Specify approved CLE label taints for each function portion

```
#pragma cle def XDLINKAGE_GET_A
{"level":"orange",
 "cdf": [
   {"remotelevel":"purple",
    "direction": "bidirectional",
    "guarddirective": { "operation": "allow"},
    "argtaints": [],
    "codtaints": ["ORANGE"],
    "rettaints": ["TAG_RESPONSE_GET_A"]
   } \
 ] }
```

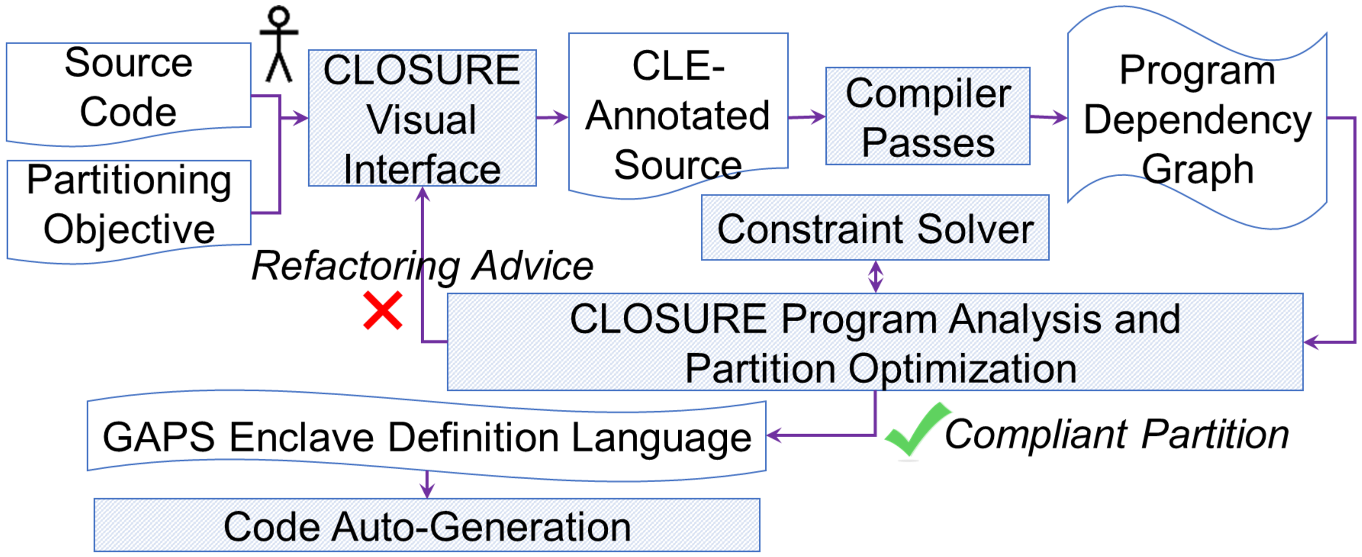
Annotate function declaration with CLE label

```
#pragma cle begin XDLINKAGE_GET_A
double get_a() {
#pragma cle end XDLINKAGE_GET_A
…
}
```


Peraton LABS

Program Analysis for Correct-by-Construction Partitions

- Assign program elements to enclaves to satisfy cross-domain constraints from annotations
- Declarative constraint solving approach
  - Efficient state-of-the-art solvers: MiniZinc and Z3
  - Declarative model easy to understand and extend
  - Model aligned with formal verification goals
  - Flexible specification of optimization objectives
- Security constraints are of three types
  - Control flow constraints
    - Cut can include only functions that are permitted to be wrapped for cross-domain RPC invocation
  - Data flow constraints
    - Cut can include only permitted taints on inputs, outputs, and return values on cross-domain RPC
  - Taint propagation constraints
    - Data flows within each enclave leading up to the cut must preserve CLE labels
    - Any security type coercion (e.g., transform of non-shareable data into redacted-shareable data) must occur through a function that has been audited and annotated by the developer



Uniform Methodology and Workflow for C, Java, and Message Flow Design Models

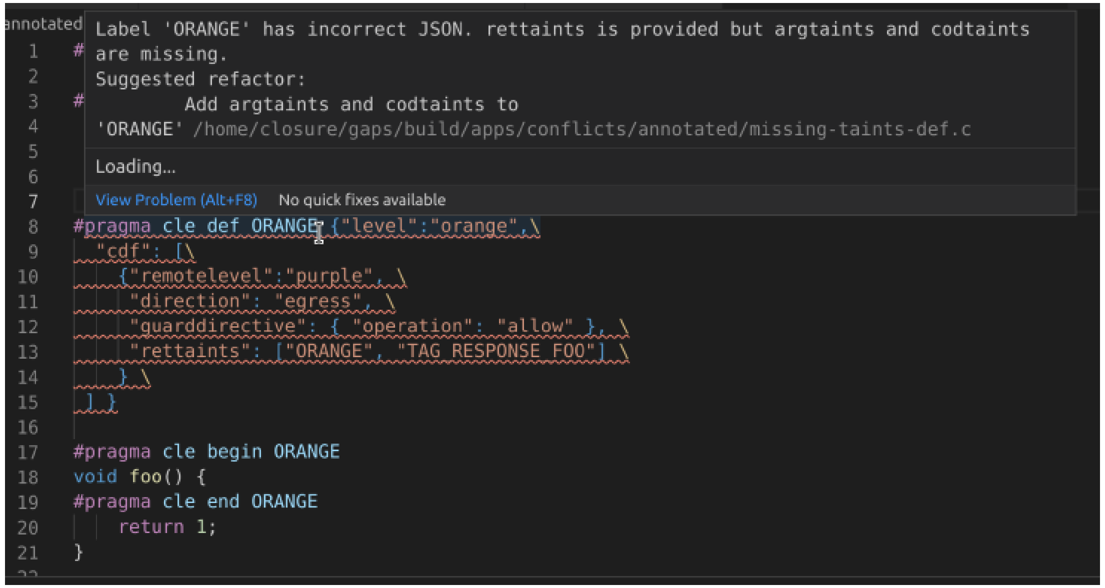
Program Complexity	SLOC	LLVM IR (B)	PDG Dot (B)	Nodes	Edges	Control Edges	Time
Example 1 (toy program)	57	13,681	35,665	96	265	75	0.089s
XDCC (useful, small program)	533	92,131	361,620	919	2,769	651	0.307s
SecDesk (real-world full web server use-case)	26,090	10,923,916	241,309,226	177,177	2,461,700	107,053	5m6.660s

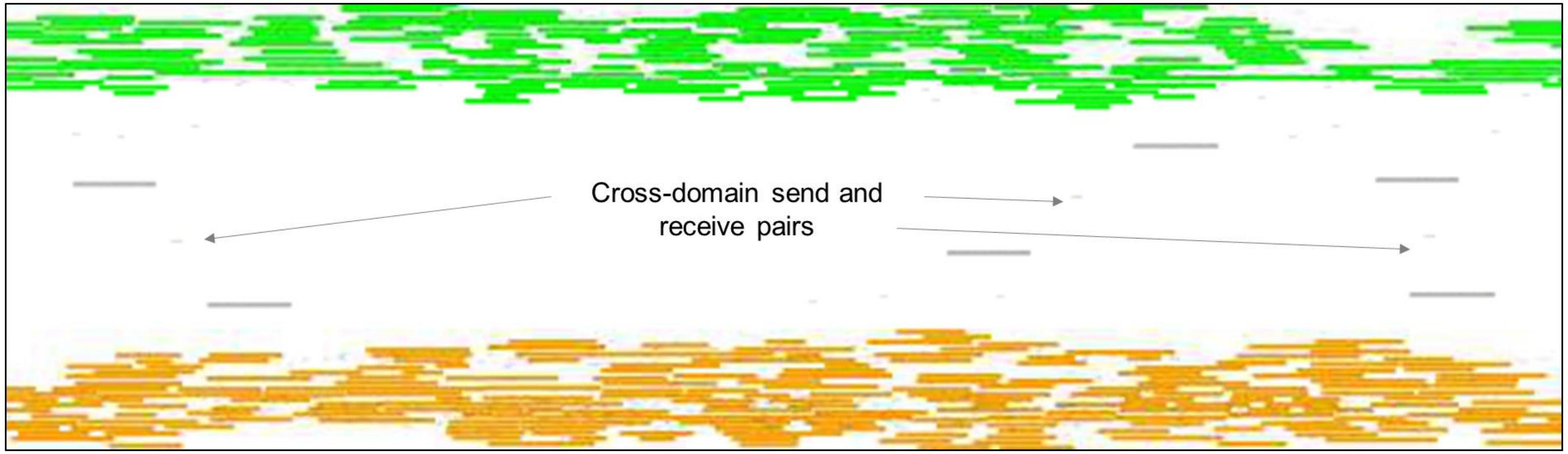
Performance of initial constraint solver based prototype

Peraton LABS

Interactive Refactoring Guidance and Diagnostics

- Provide actionable refactoring guidance when constraints are not satisfied
  - Involve developer in steps that require human input, i.e., to capture of application-semantics and information-sharing restrictions
  - Program analysis outputs used to automate code generation to avoid human errors in coding and configuration
- VSCode plugin with context-sensitive annotation hints under development



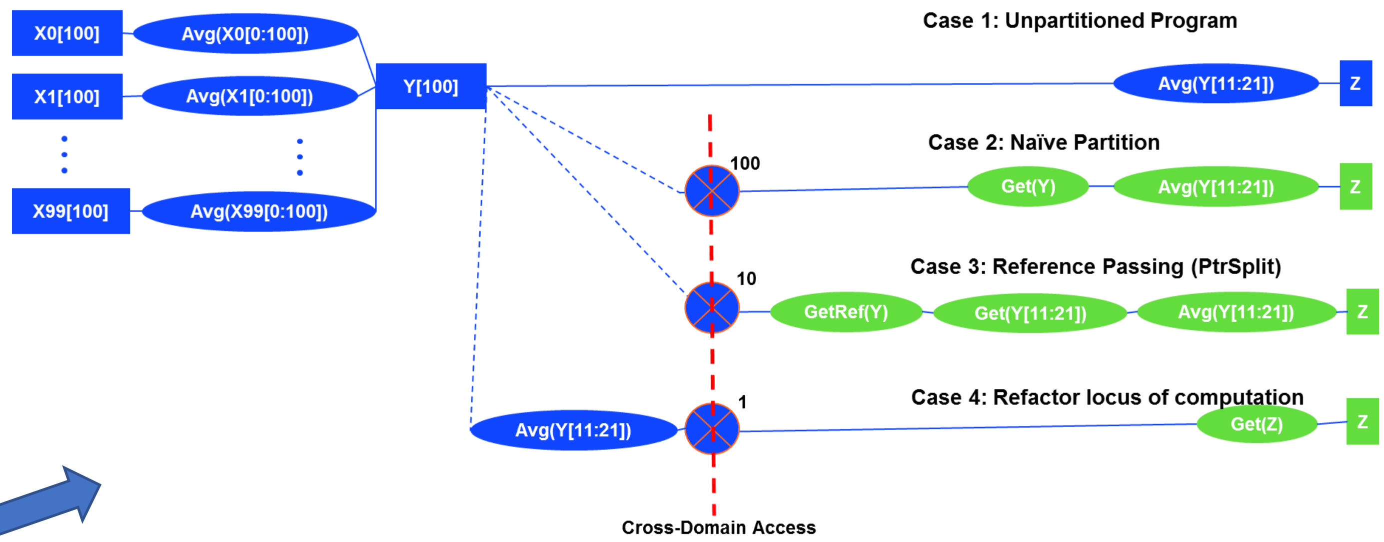


- Make it easy to visualize the cut and audit functions involved in cross-domain information sharing

Peraton LABS

Optimizing the Cross-Domain Partition

- Distributed data access is significantly more expensive than accessing data from local memory
- Heterogeneous hardware capability could require limiting functionality assigned to each enclave
- We can trade functionality across cut subject to security constraints, e.g.,
  - move average function across cut vs. passing array



Parametric optimization using integer programming to partition the wget program with budgets on: *sensitive code size* (b_c), *cross-domain flow* (b_f), *context-switch frequency* (b_s), and *interface complexity* (b_x)

	Budgets (b_c, b_f, b_s, b_x)	IP-Solving Time (s)	SCode(%)	Flow	CSwitch	Cplx	Overhead(%) (FileSize: 1M/1K) Remote	Local
①	(_*, _, _, _)	0.80	11.03	4047.0	1213.8	117.0	1493.0/6.2	13799.0/13.4
②	(50.00%, 999.0*, 38.2, _)	2.03	49.12	8.0	38.2	45.0	1.6/1.9	6.4/2.1
③	(16.00%, _, _*, _)	1.13	15.68	4052.0	198.5	137.0	412.0/7.2	1440.0/7.9
④	(_*, 2.0, 38.2, _)	1.56	78.42	2.0	38.2	14.0	1.5/2.3	7.6/3.3

Liu, S., Zeng, D., Huang, Y., Capobianco, F., McCamant, S., Jaeger, T., and Tan, G., "Program-mandering: Quantitative Privilege Separation," Accepted for presentation at ACM Conference on Computer and Communications Security (CCS), 2019

Automated Tooling for Cross-Domain Operation

Once source code has been refactored to resolve conflicts, CLOSURE auto-generates cross-domain tooling, eliminating tedious and error-prone developer effort:

1) RPC and marshaling code 2) Hardware pipelines 3) System configuration

Cross-Domain Artifacts	Description	Purpose
DFDL Data Descriptions	**Standards-based** descriptive format aligned with NCDSMO accreditation guidelines and the state of practice	Conveys payload formats to GAPS HW; **auto-generation of HW pipelines** (VHDL) generation for high-speed stream filtering
Rule Specifications	Maps GAPS tags to associated filtering and redaction rules; currently **vendor specific**	Auto-configures GAPS hardware; **working on standardization** with CDS community
Marshaling Code	Packs/unpacks in-memory data instances to fixed-size format	Formats data for parsing by the CDG
Remote Procedure Calls	Communication patterns to invoke/access data residing on remote enclaves (i.e., one-way, network fault tolerant)	Preserves intended control-flow in unpartitioned program
CLOSURE HAL Configuration	Initializes CLOSURE Hardware Abstraction Layer (HAL) with GAPS tag/device mappings for application multiplexing	Abstracts hardware concerns from application

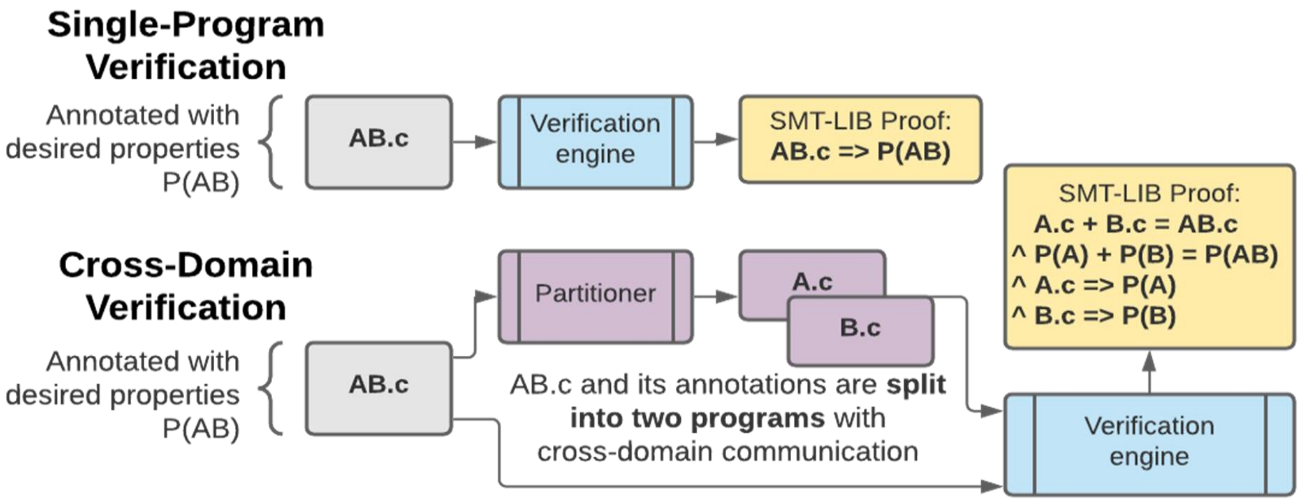
Verification in CLOSURE

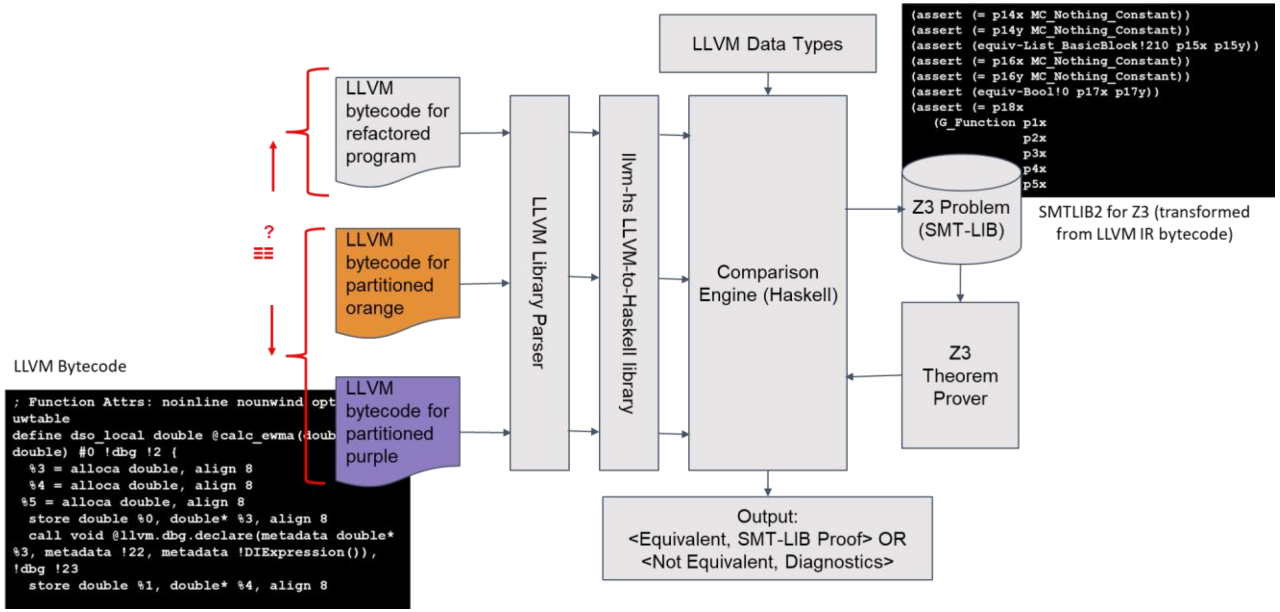
Problem: Verify that the partitioned program including any auto-generated code is:

Equivalent to the original program in behavior

and

Complies with cross-domain security constraints specified through the CLE annotations



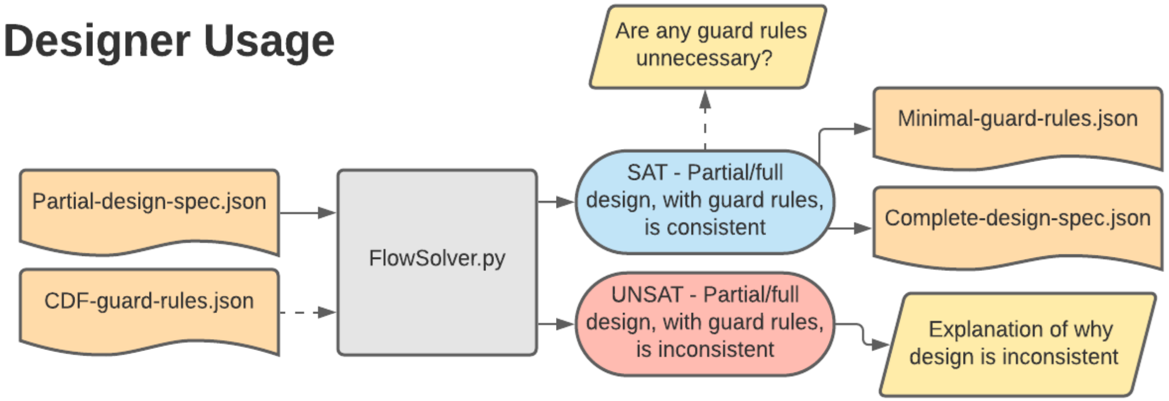


Verification engine encodes relevant program elements as constraints in SMT-LIB2 and uses the Z3 theorem prover to check that program satisfies desired properties for program equivalence and security compliance

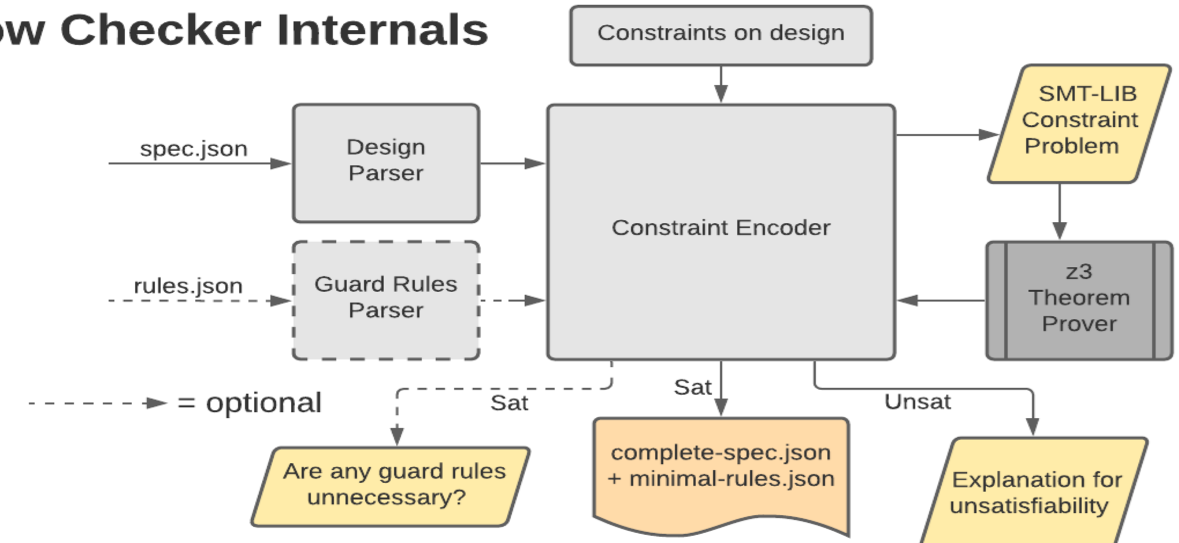
Model-Level Checker for Message Flow Specifications

Adapt same verification approach to provide model verification for message-based distributed systems

Designer Usage



Flow Checker Internals



- Check design is consistent
- Fill in missing annotations using satisfying assignments, partially specified model
- Overlay guard rules, and check they are consistent with design
- Uses Z3 theorem prover to formally check design translated to SMT-LIB constraints
- Modular, can be extended to include more constraints in the future

```
python3 FlowSolver.py partial/case1_valid.json
--rules rules/case1.json
```


"flows": [
{"id": 1201,"msg":...},
{"id": 1301,"msg":...},
{"id": 1401,"msg":...},
{"id": 1501,"msg":...},

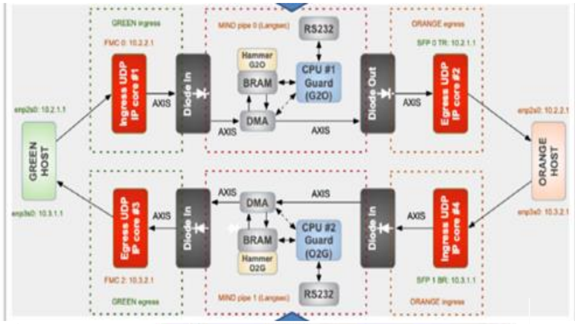
Consistency Checked, Missing Labels Filled In

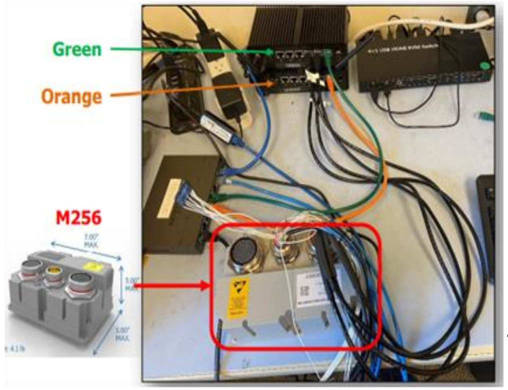
"flows": [
{"id": 1201,"msg":...,"label":"ALLOW_ORANGE_ORANGE"},
{"id": 1301,"msg":...,"label":"ALLOW_ORANGE_ORANGE"},
{"id": 1401,"msg":...,"label":"ALLOW_ORANGE_GREEN"},
{"id": 1501,"msg":...,"label":"ALLOW_ORANGE_ORANGE"},

Peraton LABS

GAPS Hardware

Monitoring & Inspection Device (MIND)

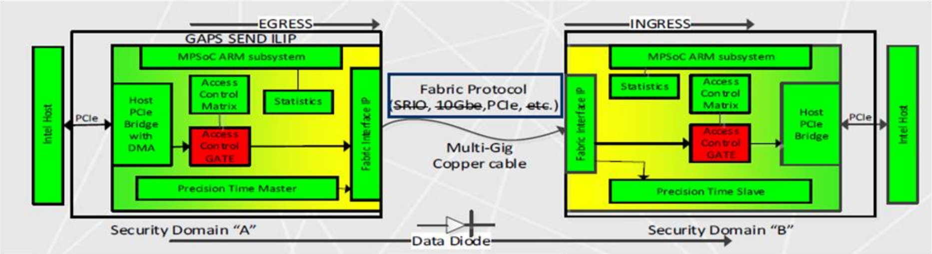


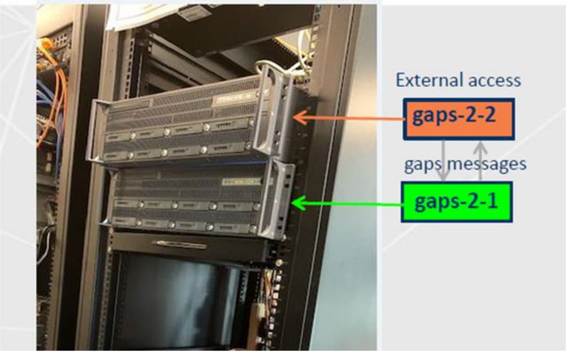


Source: GE Research

- Ethernet-based, bump-in-the-wire
- Payload parsing/redaction in VHDL
- Isolated Forward/reverse pipelines
- Xilinx and GE avionics M256 form-factors

InLine Interface Processor (ILIP)

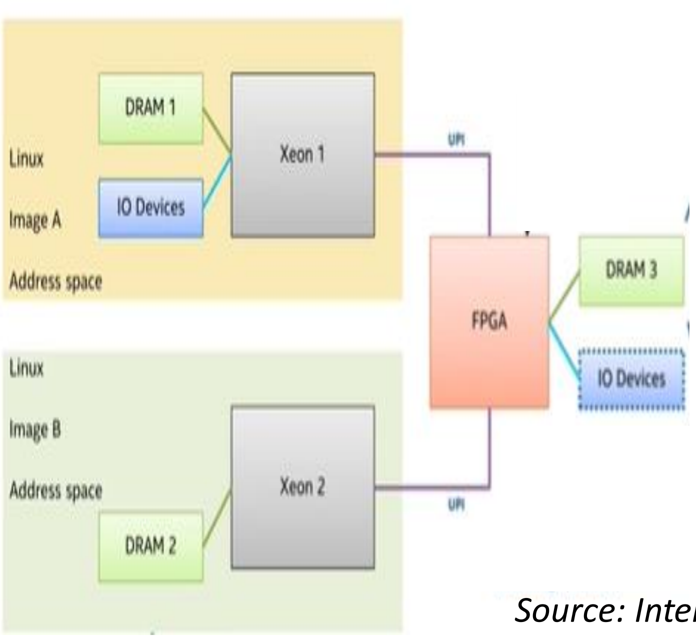




Source: Mercury Systems

- PCIe-based, Xilinx MPSoC bookends
- Segmentation/Reassembly for large payloads (1 MB+, theoretical 1 GB)
- Redaction guided by payload offsets

Extended Secure Capabilities Architecture Platform and Evaluation (ESCAPE)



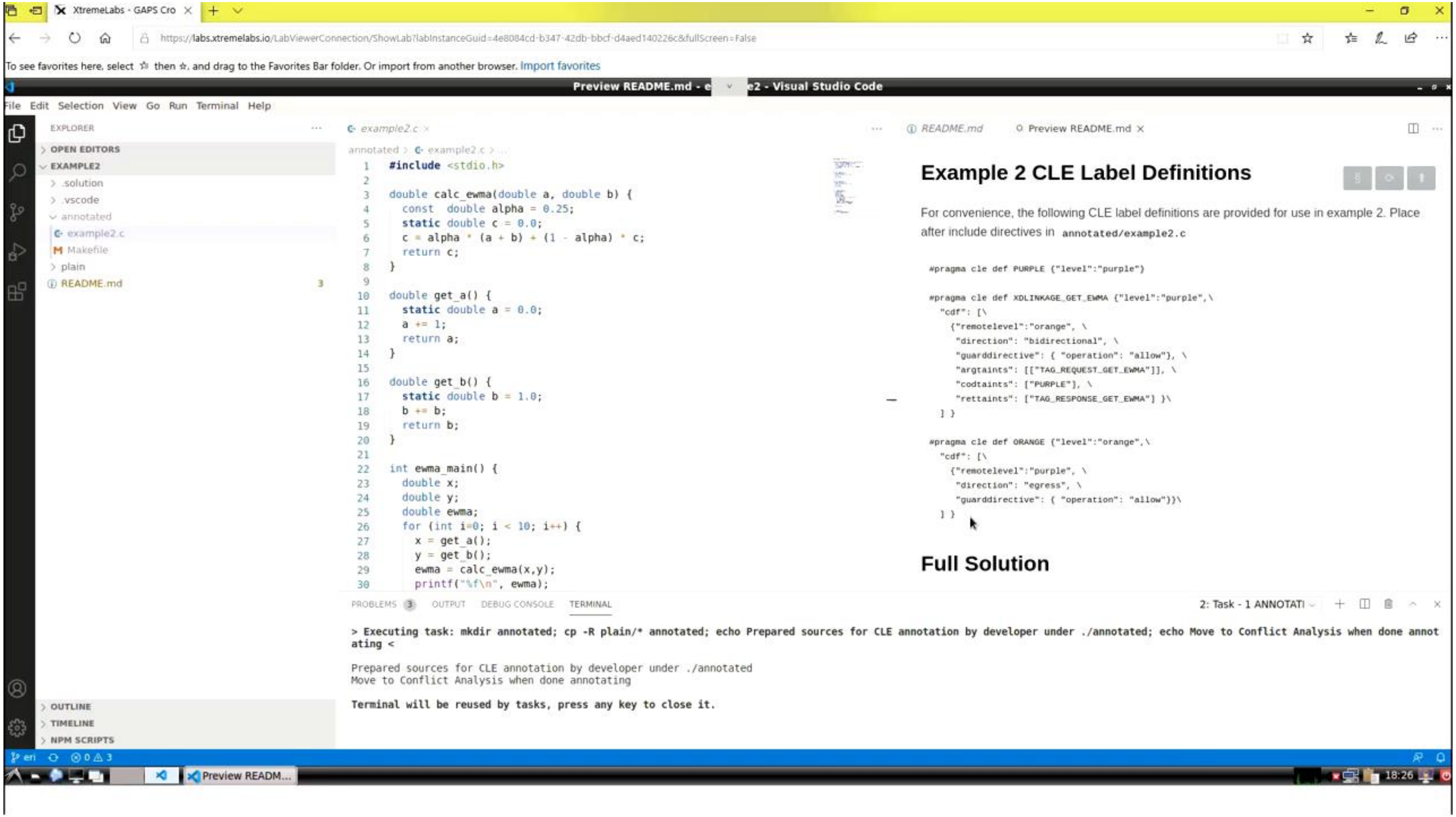
Source: Intel

- 2-Xeon CPUs connected over UPI to FPGA
- Address-filtering to allow or disallow read/writes to shared memory pool
- UPI transfer speeds up to 10.4 GT/s

CLOSURE provides cross-domain applications with uniform API abstractions across diverse GAPS hardware with multiple link technologies and performance characteristics – network systems, backplane buses, and chip-to-chip interconnects

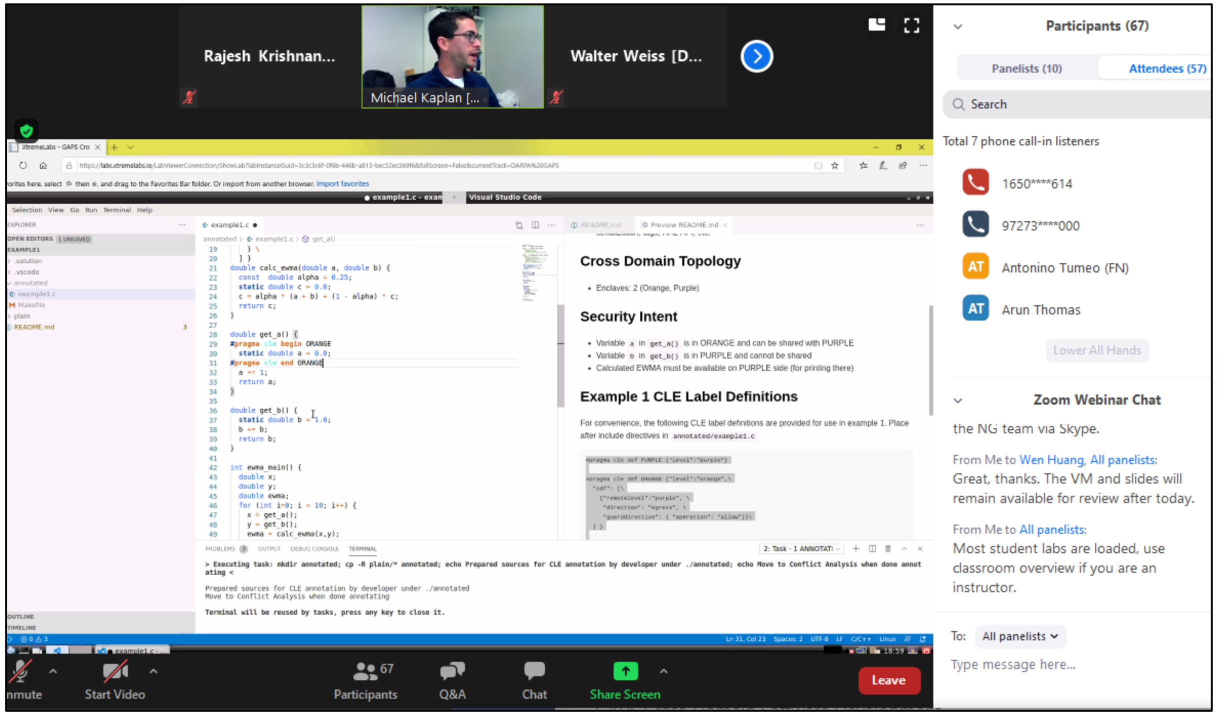
Peraton LABS

Demonstration



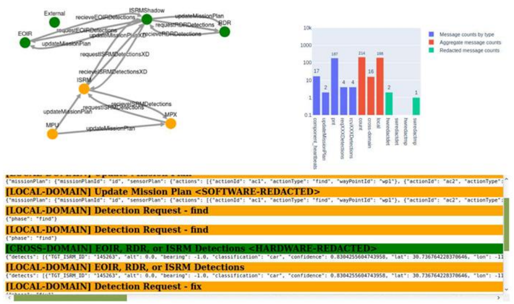
Peraton LABS

Program Highlights

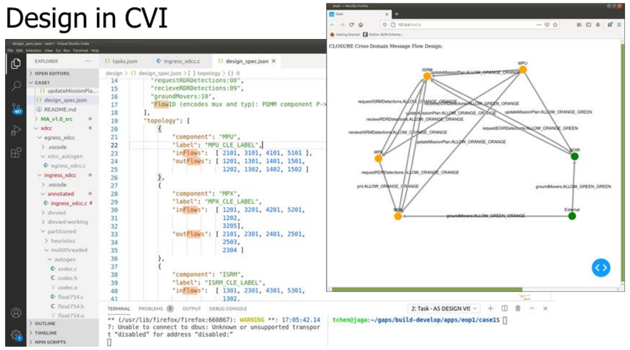


Virtual Interactive Workshop
at 2020 ERI Summit

TA3 Mission Application Demonstration

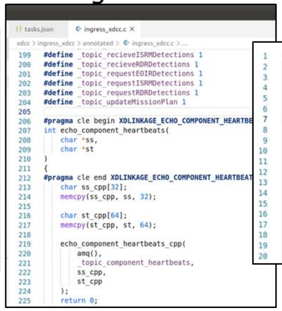


Design in CVI



Large image transfer with redaction

Auto-generated Annotated Code



Auto-generated DFDL

End of Phase 1 Demonstrations (Feb 2021)
NGC AMQP-based Mission App, Message-Flow Partitioning, Large Image Transactions with Meta-Data Redaction

<u>Other Highlights</u>

- Briefings to multiple transition partners and PoR
- GAPS posters and demos to be featured at ERI 2021

Metrics

	6-Mo. Demo 04/2020	ERI 08/2020	EOP1 02/2021	EOP2 2022
	Full Workflow, Integrated Demonstration	**Usable Toolchain, Automation**	**Capstone Integrated Demonstration**	**2+ Languages 3+ Enclaves Real-time Concurrent**
Message Types	<5	<5	<10	<100
App Processes	1	1	6	20
# of Enclaves	2	2	2	3+
Auto-Leverage:	0	.95	1.0	
Input LoC	C++: 336	C: 25,000	C++: 18,500	<100,000
SLoC Changes	444	77	0	
Languages	Limited C++	C	Control Flow: C, Limited C++ Message Flow: Any	Additional modern language support
Partitioning Style	Control Flow	Control Flow	Control & Message Flow (independently handled)	Control & Message Flow (jointly handled)
RPC Style	Asynchronous	Synchronous	Async/one-way, Synchronous	Robust distributed, concurrency
RPC Generation	Manual	Automatic	Automatic	Automatic with additional CLE parameterization
Deployment Linux Distro	Server Ubuntu 19.10	Server, VM Ubuntu 19.10	Server, VM, Docker Ubuntu 20.04, Centos 7.8	Server, VM, Docker Ubuntu, Centos, EOP2 specific distros

Themes for CLOSURE Metrics

- Performance
  - Size and complexity of programs handled (time required to analyze/verify)
  - Optimization (of the cut) and end-to-end performance (engineering in later phases)
- Language Expressiveness and Portability
  - Coverage within and across languages
  - Target architectures supported
- Developer Productivity
  - How much developer effort can we reduce (auto-generation leverage)
  - Ease of adoption (ERI, use of VSCode, Docker, open source)
- Transition
  - Standardization & accreditation
  - Insertion into real missions

Lessons Learned

- Physical isolation is key for protections against chip-level attacks (e.g., Spectre)
- Transition depends on a balance between innovations (e.g., CLOSURE) and community standards for interoperability and accreditation (e.g., DFDL, RTB)
- Surgically add innovations into established technologies (e.g., LLVM, Z3, VSCode)
- Right abstractions (e.g., PDG) and technologies (e.g., constraint solvers) makes it easier to generalize the solution (across languages, architectures, application models)
- Interactive feedback and auto-generation are both critical to developer experience
- Integrating with hardware vendors early and often reduced risk for the program
- End-to-end testing (in emulator) facilitates seamless deployment on hardware
- Gain design insights by observing novice users of your system

Conclusions

CLOSURE solves the problem of automated correct-by-construction partitioning of programs, and our open-source toolchain simplifies cross-domain systems development.

Thank you for the opportunity to speak today, we look forward to possible collaboration in the future.

GAPS is available on GitHub: https://gaps-closure.github.io/

CLOSURE Team

- Peraton Labs
  - Robert Brotzman
  - Ta Chen
  - Andrzej Cichocki
  - Ben Flin
  - Mike Kaplan
  - Rajesh Krishnan
  - Tony McAuley

Leads All Tasks
Reports to DARPA
Interfaces to TA1/TA3

Prior Work: SHARE, DADC, SQATool, FCS, Trailblazer, Chameleon

Capabilities: cross-domain solution design and development, control-flow analyses, constraint solvers, code morphing

- Penn State
  - Prof. Gang Tan
  - Prof. Trent Jaeger

Pointer Analyses & Privilege Separation

Prior Work: PTRSplit, Program Mandering

Capabilities: selective bounds checking, parameter trees, PDGs

- UPenn
  - Prof. Boon Thau Loo
  - Dr. Nik Sultana

Program Decomposition & Resource Allocation

Prior Work: DeDOS, Chopflow

Capabilities: program splitting, legacy binary support, multi-target orchestration, runtime environments

- Columbia
  - Prof. Stephen Edwards

Compiler, Language Theory & Concurrency

Prior Work: SHIM

Capabilities: compiling parallel algorithms, efficient programming across the software/hardware boundary, language/compiler extensions for embedded systems

Q&A

Contacts

DARPA:
GAPS@darpa.mil

Peraton Labs:
Michael Kaplan
mkaplan@peratonlabs.com

Rajesh Krishnan
rkrishnan@peratonlabs.com

Software

https://gaps-closure.github.io
https://github.com/gaps-closure

Publications

[1] "Fine-grained Program Partitioning for Security."Huang, Z.; Jaeger, T.; and Tan, G. In 14th European Workshop on Systems Security (EuroSec), pages 21–26, 2021.

[2] "Lightweight Kernel Isolation with Virtualization and VM Functions.“ Narayanan, V.; Huang, Y.; Tan, G.; Jaeger, T.; and Burtsev, A. 16th ACM International Conference on Virtual Execution Environments (VEE),157–171. 2020.

[3] "Program-mandering: Quantitative Privilege Separation.“ Liu, S.; Zeng, D.; Huang, Y.; Capobianco, F.; McCamant, S.; Jaeger, T.; and Tan, G. In 26th ACM Conference on Computer and Communications Security (CCS), pages 1023–1040, 2019

[4] "Flightplan: Dataplane Disaggregation and Placement for P4 Programs.“ Nik Sultana, John Sonchack, Hans Giesen, Isaac Pedisich, Zhaoyang Han, Nishanth Shyamkumar, Shivani Burad, André DeHon, and Boon Thau Loo, University of Pennsylvania. Published at NSDI'21

[5] "Leveraging In-Network Application Awareness“ Nik Sultana. Published at NAI'21

[6] "Debugging strongly-compartmentalized distributed systems.“ Henry Zhu, Nik Sultana, Boon Thau Loo. Published at APDCM'21.

[7] "Meta-level issues in Offloading: Scoping, Composition, Development, and their Automation.“ André DeHon, Hans Giesen, Nik Sultana, Yuanlong Xiao. Published at LATTE'21

[8] "FDP: A Teaching and Demonstration Platform for Networking“ Heena Nagda, Rakesh Nagda, Swapneel Sheth, Nik Sultana, Boon Thau Loo (Abstract demo) Published at SIGCSE'21

[9] "Demo: Disaggregated Dataplanes“ Heena Nagda, Rakesh Nagda, Nik Sultana, Boon Thau Loo Published at ICDCS'21

Peraton LABS