



Title	Final Report
Date	May 7, 2024
Contract	HR001119C0106
Program	CLOSURE
Technical Area	TA2
Lead Organization	Peraton Labs
Technical Points of Contact Mr. Michael A. Kaplan 150 Mt. Airy Road Basking Ridge, NJ 07920 (908) 748-2805 mkaplan@peratonlabs.com	Administrative Point of Contact Ms. Josephine Micallef 150 Mt. Airy Road Basking Ridge, NJ 07920 (908) 748-2599 jmicallef@peratonlabs.com

This report features embedded documents best viewed in Adobe Acrobat including the C/Java/C++ Handbooks and relevant project presentations.

DoD Distribution Statement A – Approved for Public Release. Distribution is unlimited. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR0011-19-C-0106. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA).

REPORT DOCUMENTATION PAGE						
1. REPORT DATE 20240507		2. REPORT TYPE Final Report		3. DATES COVERED <table border="1"> <tr> <td>START DATE 09/11/2019</td> <td>END DATE 05/23/2024</td> </tr> </table>	START DATE 09/11/2019	END DATE 05/23/2024
START DATE 09/11/2019	END DATE 05/23/2024					
4. TITLE AND SUBTITLE CLOSURE						
5a. CONTRACT NUMBER HR0011-19-C-0106		5b. GRANT NUMBER BAA HR001119S0017		5c. PROGRAM ELEMENT NUMBER N/A		
5d. PROJECT NUMBER N/A		5e. TASK NUMBER CLIN 0003		5f. WORK UNIT NUMBER N/A		
6. AUTHOR(S) Mr. Michael A. Kaplan						
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Peraton Labs (formerly Perspecta Labs Inc.) 150 Mt. Airy Rd, Basking Ridge, NJ 07920				8. PERFORMING ORGANIZATION REPORT NUMBER N/A		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Naval Information Warfare Center Pacific Robert Wendland 53560 Hull Street San Diego CA 92152-5001			10. SPONSOR/MONITOR'S ACRONYM(S) NIWC	11. SPONSOR/MONITOR'S REPORT NUMBER(S) N/A		
12. DISTRIBUTION/AVAILABILITY STATEMENT DoD Distribution Statement A – Approved for Public Release. Distribution is unlimited. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR0011-19-C-0106. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA).						
13. SUPPLEMENTARY NOTES N/A						
14. ABSTRACT Peraton Labs developed CLOSURE (Cross-domain Language-extensions for Optimal SecUre Refactoring and Execution) to address the challenges associated with building cross-domain applications in software. CLOSURE extends existing programming languages by enabling developers the ability to express security intent through overlay annotations and security policies such that an application can be compiled to separate binaries for concurrent execution on physically isolated platforms.						
15. SUBJECT TERMS cross-domain systems, program partitioning, language extensions, constraint modeling, correct-by-construction methodologies						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES		
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U	UU	14 + embedded attachments		
19a. NAME OF RESPONSIBLE PERSON Robert Wendland		19b. PHONE NUMBER (Include area code) 619-553-5913				

Table of Contents

I. Introduction	4
II. Task Objectives	5
III. Technical Problems and Challenges	6
IV. General Methodology	7
V. Technical Results.....	7
VI. Important Findings and Conclusions	12
VII. Significant Hardware Development	12
VIII. Special Comments	12
IX. Implications for Further Research	12
X. Other Items of Interest	13
XI. Detailed Technical Documentation	13

I. Introduction

DARPA's Guaranteed Architecture for Physical Systems (GAPS) is a research program that addresses software and hardware for compartmentalized applications where multiple parties, each with strong physical isolation of their computational environment, have specific constraints on the sharing of data (possibly including redaction requirements) with other parties, and any data exchange between the parties is mediated through a guard that enforces the security requirements.

Peraton Labs' Cross-domain Language extensions for Optimal SecUre Refactoring and Execution (CLOSURE) project is building a toolchain to support the development, refactoring, and correct-by-construction partitioning of applications and configuration of the guards. Using the CLOSURE approach and toolchain, developers will express security intent through annotations applied to the program, which drive the program analysis, partitioning, and code auto-generation required by a GAPS application.

Problem: The machinery required to verifiably and securely establish communication between cross-domain systems (CDS) without jeopardizing data spillage is too complex to implement for many software platforms where such communication would otherwise be desired. To regulate data exchanges between domains, network architects rely on several risk mitigation strategies including human fusion of data, data-diodes, and hypervisors which are insufficient for future commercial and government needs as they are high overhead, customized to specific setups, prone to misconfiguration, and vulnerable to software/hardware security flaws. To streamline the design, development, and deployment of provably secure CDSs, new hardware and software co-design tools are needed to more effectively build cross-domain support directly into applications and associated hardware early in the development lifecycle.

Solution: Peraton Labs developed CLOSURE (Cross-domain Language-extensions for Optimal SecUre Refactoring and Execution) to address the challenges associated with building cross-domain applications in software. CLOSURE extends existing programming languages by enabling developers the ability to express security intent through overlay annotations and security policies such that an application can be compiled to separate binaries for concurrent execution on physically isolated platforms.

The CLOSURE compiler toolchain interprets annotation directives and performs program analysis of the annotated program and produces a correct-by-construction partition if feasible. CLOSURE automatically generates and inserts serialization, marshalling, and remote-procedure call code for cross-domain interactions between the program partitions.

In this document, we summarize the results of the CLOSURE project at the end of Phase 3 in the format specified by the contract. Key objectives including support for two languages (C and Java) and three enclaves were met. In addition, we now have preliminary support for the C++ language. For convenience, we have consolidated the deliverables into a software release and a user handbook each for each language:

- C Documentation: <https://gaps-closure.github.io/cdoc.pdf>
- C Software Release: <https://github.com/gaps-closure/build/releases/tag/v3.0>
- Java Documentation: <https://gaps-closure.github.io/jdoc.pdf>
- Java Software Release: <https://github.com/gaps-closure/java-closure>
- C++ Documentation: <https://gaps-closure.github.io/cppdoc.pdf>
- C++ Software Release: <https://github.com/gaps-closure/cpp-closure>

This report features embedded documents best viewed in Adobe Acrobat including the C/Java/C++ Handbooks and relevant project presentations.

II. Task Objectives

Key task-level objectives, relevant components, and supporting materials are summarized in the table below along with references to their respective report sections:

Task	Relevant CLOSURE Tools/Results	Supporting Materials
Compiler Tools	MULES, CAPO	C: Handbook §3.1, §3.2, §3.3 Java: Handbook §3.1, §3.2, §3.3 C++: Handbook §3.1, §3.2, §3.3
Multi-Target Binary Generation	CLOSURE generated binaries have been tested on x86, ARM64, A53, and MicroBlaze CPUs. Binary generation not applicable to Java since JVM is multi-platform.	C: Handbook §3.7, §3.10 (examples tested in X86 and ARM QEMUs) <u>Presentation</u> : GE Edge Symposium 2022, (A53/MicroBlaze using Xilinx ZCU102+)
User Tools	Closure Visual Interface (CVI) based on VSCode. Includes CLOSURE Language Extensions (CLE) plugin.	C: Handbook §3.8 C++/Java: same VSCode-based CVI
Transition and Integration Support	• oneAPI talk and demonstration (with Intel) • C demonstration on ZCU102 (with GE) • Technical Interchange Meeting 3 (TIM3) held at Peraton Labs (with Northrop Grumman and GAPS performers) • US Army DEVCOM briefing with detailing CLOSURE/X-ARBITOR synergies (jointly with Arcfield) • ATAK Sync Application over X-ARBITOR (with Arcfield) • Analysis of Partner Use-Case Surrogate	• <u>Presentation</u>: Annotation-Driven Analysis and Partitioning of High-Performance Cross-Domain Applications (attached) • EoP2 ZCU102 Github • EoP2 Java Application Github • TIM3 briefing (attached), Dockerized release of verification tools • <u>Presentation</u>: Technology Enhancements for Agile Cross-Domain Systems (attached) • <u>Report</u>: CLOSURE Integration with Accredited CDS (X-ARBITOR) for DARPA GAPS (see DARPA vault) • <u>Presentation</u>: Surrogate Application Design, Partitioning Concept, and Message Types (see DARPA vault)
Continuous Integration	Develop branch maintained using GitHub CI/CD tools. Latest and stable Docker images periodically generated.	Accessible via: https://github.com/gaps-closure https://hub.docker.com/search?q=gapsclosure
Source Code	CLOSURE project is open-source	https://github.com/gaps-closure
Annotations	CLOSURE Language Extensions (CLE)	C: Handbook §3.1 Java: Handbook §3.1 CLE Schema on GitHub C++: Handbook §3.1
Emulator	EMU	C: Handbook §3.7 https://github.com/gaps-closure/emu
Technical Papers	Conference Publications by CLOSURE team members	Full list of references in §XIXI.
Task Test Plans	End-to-end evaluation of CLOSURE for C and Java was done using a series of example programs. This set can continue to be used for regression testing. For C++ we developed a suite of simple test programs that exercise specific language features we supported.	C: Handbook §3.10 C++: Handbook: §3.8 C ERI Summit Examples 1-3 C/C++ EoP1 Cases 1-3 (see EoP1 reports) Java EoP2 3-Enclave Application C++ Examples
Benchmarking Analysis	• We used the HAL daemon to benchmark performance of C and Java programs over Emulated, ILIP and MIND-DMA GAPS hardware. • Additionally, we used a higher performance HAL library to benchmark	• Phase 1 final briefing (HAL daemon performance) • C: Handbook §3.5 describes the HAL implementations and benchmarking. • C: Handbook §5.9 describes ESCAPE benchmarking results.

	performance of C programs over ESCAPE-Shared-Memory, MIND-DMA and X-ARBITOR. • For the ESCAPE testbed, we identified fundamental limitations of shared-memory performance. • CLOSURE conflict analyzer and verifier evaluated over a range of programs.	• Conflict analyzer and Verifier performance reported in Peraton EoP1 Briefing available on GAPS Confluence.
--	---	--

III. Technical Problems and Challenges

The key technical challenges and problems for the TA2 GAPS effort and CLOSURE's approach towards addressing them are listed in the following table:

TA2 Challenge/Problem	CLOSURE Approach	Supporting Materials
<u>Expression:</u> How to design intuitive language extensions to specify cross-domain security intent	CLOSURE Language Extensions (CLE) describe cross-domain policies using overlay annotations. C and Java use the same CLE JSON syntax, and apply the annotations using pragmas or Java annotations respectively to associate the policy with sensitive data and functions. Our C expression methodology carried over to C++ with additional extensions for object oriented constructs such as classes, fields, and methods.	C: Handbook §3.1, §3.8 Java: Handbook §3.1 C++: Handbook §3.1 CLE Schema on GitHub
<u>Program Analysis and Guided Refactoring:</u> How to identify conflicts in annotated programs and provide actionable refactoring	CLOSURE models the control and data dependencies of the program using a Program Dependency Graph (PDG) and Static Value Flow (SVF) Points-To Analysis, and inputs this program model along with the annotated policy to a constraint-solver to identify conflicts. PSU's PDG and SVF , and KIT's JOANA support this dependency analysis for C/C++, and Java respectively. CLE plugin provides guided feedback for conflicts.	C: Handbook §3.1, §3.2, §3.3 Java: Handbook §3.1, §3.2, §3.3 C++: Handbook §3.1, §3.2, §3.3 CLE Plugin described in C Handbook §3.8
<u>Automation:</u> How to reduce developer effort via automated code generation, refactoring, and end-to-end optimization	CLOSURE reduces developer effort by auto-generating the cross-domain tooling required to make a refactored (policy-compliant) program ready for compilation. Cross-Domain RPC logic for marshalling and serialization of data between enclaves (.c/.h for C, Aspects for Java, and .cpp/.h for C++), DFDL data description files, and Hardware Abstraction Layer (HAL) API calls and configurations are auto-generated, eliminating significant implementation cognitive demands on the application developer.	C: Handbook §3.4, §3.5 Java: Handbook §3.3 C++: Handbook §3.5, §3.6
<u>Verification:</u> How to argue formal correctness and compliance of partitioned programs	CLOSURE leverages constraint modeling and its ParTV tool to enforce the following guarantees: consistent annotation, satisfies CLE constraint model, functional equivalence (partitioned v. unpartitioned), preservation of annotations in partitioned code, and constraint validation of partitioned code	<u>Presentation:</u> GAPS VV&A Discussion: Focus on CLOSURE Technical Interchange Meeting #3 (attached)
<u>Ecosystem:</u> How to foster adoption and participation from application, toolchain, and hardware developers.	CLOSURE is built upon established technologies including VSCode, LLVM, Clang, QEMU and others to ease transition to early adopters.	<u>Poster:</u> ERI Summit 2021 (attached) <u>Presentation:</u> Technology

	Supports standardization of guard rules and configuration using DFDL/DAGR/DFTVL.	Enhancements for Agile Cross-Domain Systems (attached) <u>Presentation</u> : An early preview of DFDL-Aware Guard Rules (attached)
--	--	---

IV. General Methodology

The CLOSURE Methodology follows the workflow shown in Figure 1. The same workflow supports C, Java, and C++ toolchains with minor substitutions for language-specific tools where needed. The stages of the workflow are as follows: 1) Annotation-driven development for correct-by-construction partitions with interactive feedback for guided refactoring, 2) Automated generation of cross-domain artifacts, compilation, and verification of partitioned program, 3) Seamless support for four heterogeneous Cross Domain Systems consisting of legacy or GAPS hardware architectures, and 4) Emulation for pre- deployment testing. Detailed discussion of CLOSURE methodology and workflow available in C Handbook §1, Java Handbook §1, and C++ Handbook §1.

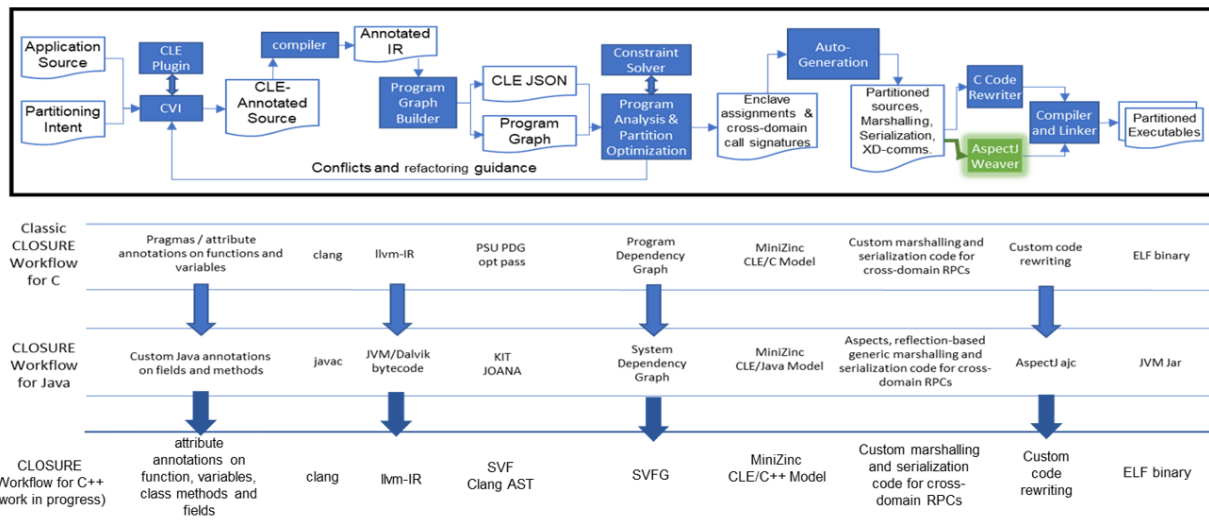


Figure 1: CLE and CLOSURE workflow maps to multiple programming languages with the use of language-specific models and tools

V. Technical Results

The key technical results from the GAPS CLOSURE project are listed below. In summary, we applied our methodology to 4 different languages (C, Java, C++, and DPC++), as well as embedded-C (limited C subset for compiling to embedded platforms such as ZCU102+ boards). Additionally, we supported 4 CDS platforms (GE MIND, Mercury ILIP, Intel ESCAPE, and AFRL X-ARBITOR).

CLOSURE Toolchain for C: The CLOSURE toolchain for C was our most mature toolchain throughout the program, key results included:

- We formalized the CLOSURE constraint model for CLE-annotated C programs and implemented it in MiniZinc. This allowed us to replace the somewhat ad-hoc graph

clustering algorithm used in Phase 1 with a more rigorous and cleanly-specified constraint-solver based approach in Phase 2 [C Handbook §3.2].

- We integrated enhancements developed by our team member UPenn to make the auto-generated RPC code more reliable to delays, losses, and restarts; in addition, we extended CLE to be able to express the preferred RPC style and function constraints such as idempotence that affect what reliability mechanisms can be used [C Handbook §5.10]
- We have integrated the ParTV [5] post-partition verification tool developed by our team member Columbia University, so that partitioned programs are checked for program behavior equivalence with the unpartitioned (subject to some assumptions about the RPC code), and for complying with the security annotations in the original unpartitioned code, taking into account the additional auto-generated code. Phase 2 performance evaluation of ParTV shown in Figure 2, detailed in [5].
- We have developed a CLE plugin for CVI (in VSCode) that aids the developer with completion hints when defining and applying annotations, visualizing the feasible partition determined by the conflict analyzer if successful, and for providing contextual refactoring guidance based on the minimum unsatisfiable set of constraints identified by the conflict analyzer.
- In collaboration with our team member, Penn State University, we have developed a Program Dependence Graph abstraction and builder tool for C++, which is the first step toward applying and extending CLOSURE to the C++ language.

CLOSURE Toolchain for JAVA: In phase 2 we added Java as the second language supported by CLOSURE and demonstrated using 3-enclave scenarios. Significant results within this topic include:

- We have adapted the CLOSURE Language Extensions annotation syntax and methodology for use with Java. We use the same CLE-JSON schema as with the C toolchain, but we define and use Java-style annotations (vs. pragmas for C programs). The annotations are applied to methods and fields and are propagated by the Java compiler to the JVM bytecode, which we use for program analysis.
- We apply the System Dependency Graph abstraction and the JOANA tool from Karlsruhe Institute of Technology, which we modified to support Java (CLE) annotations (vs. only JOANA annotations).
- We formalized the CLOSURE constraint model for CLE-annotated programs in a subset of the Java language, and implemented the model using MiniZinc.
- We applied Aspect-Oriented Programming to handle cross-domain serialization, RPC, marshalling at the inter-enclave cut and isolation for intra-enclave program elements.
- We developed a demonstration application in Java that is partitioned using CLOSURE into three enclaves: a camera controller, a high-side webserver, and a low-side webserver.

Program	SLoC	Globals	Target partition
Toy example	37	8	Access only weighted avg
chsh	564	18	Protect password file
passwd	1168	47	Protect password file
useradd	2395	73	Protect password file
thttpd	11403	273	Separate <i>send()</i> , <i>send2()</i>
Security Desk	23654	1341	Isolate facial recognition

Fig. 10. A summary of the programs used for our evaluation.

Program	Size (SLoC)	Annotations		RPC code (SLoC)
		Function	Data	
Toy example	37	1	3	1064
chsh	564	1	2	1054
passwd	1168	2	2	1111
useradd	2395	2	3	1176
thttpd	11403	3	2	1329
Security Desk	23654	7	6	1646

Fig. 11. The number of user-provided annotations to express a security policy

Experiment	Total time	Conflict Analysis	Code Gen.	Verification
Toy example	9.9 s	1.3	5.0	3.6
chsh	14	3.4	4.4	5.7
passwd	18	3.6	6.1	8.1
useradd	29	7.7	7.6	14
thttpd	1390	620	390	380
SecDesk	1560	630	430	500

Fig. 12. Total time taken by the partitioner in each phase.

Figure 2: Published Results from ParTV Study [5]

- We developed a proof-of-concept application of CLOSURE to a small Android app.
- The mainstream program analysis tools, WALA and SOOT were unable to handle large programs such as the Signal Android app. In order to overcome the model-building limitations, we developed a custom approach based on disassembling to Smali file and post-processing to build a partial Program Dependency Graph model. The model will require additional work to handle a larger subset of the Java language.
- We identified limitations of annotation-driven program analysis and partitioning for Java. First, some Java constructs such as lambdas are fundamentally not amenable to annotation-driven analysis. Second, the structure of the program with dense connectivity between classes/packages, as was the case with the Signal Android app, was not conducive to functional partitioning where a sparse cut is preferable. Further work is needed to provide a more complete solution for the Java language.

Prototype CLOSURE Toolchain for C++: Late in Phase 3 we were tasked to develop a C++ toolchain in support of transition partner use-cases. We achieved tools for the expression, annotation, conflict analysis, and division steps of the workflow, with manual representations of downstream artifacts such as RPCs, DFDL, and guard configurations.

- Developed a novel method for annotation utilize Clang features in C++20.
- Extended CLE to C++ object-oriented constructs such as classes, fields, and methods.
- Developed a constraint model for C++ primitives and basic CDS policies.
- Redesigned and implemented a proper language-aware code divider based on the Clang AST.
- Tested small programs with specific class/object unit tests to exercise the constraints and developed tools.
- Toolchain prototype is easily extensible to additional C++ features which can be incrementally added per the needs of the target applications.

Joint Toolchain using Intel oneAPI: In collaboration with Intel, we combined the CLOSURE toolchain and the Intel oneAPI toolchain to address annotation-driven partitioning of high-performance cross-domain applications that can benefit GPU offload:

- See attached presentation “Annotation-Driven Analysis and Partitioning of High-Performance Cross-Domain Applications.” [APSP app available on GitHub](#).
- This work was presented at the Intel Federal Summit on June 22, 2022.

CLOSURE Support for Four Heterogeneous CDSs: In phase 2 a second HAL implementation was created to complement the flexible middleware HAL-Daemon. While the HAL-Daemon communicated with applications using ZMQ the new HAL-Library implementation linked directly to the partitioned applications, allowing for higher-performance cross-domain communication. Both HAL implementations support the same HAL cross-domain communication (XDCOMMS) API calls, allowing both implementations to run the same CLOSURE compiled code, with no change to the rest of the CLOSURE toolchain. By the end of Phase 3, CLOSURE has been used with four heterogeneous types of communication channel: PCIe bus-based (Mercury ILIP GAPS CDS), DMA-based (GE MIND GAP CDS), shared-memory-based (Intel GAPS CDS) and File-based (X-ARBITOR CDS).

- In Phase 3 we extended the HAL library to support communication using shared-memory, which allowed cross-domain communication among CLOSURE partitioned binaries using

Intel ESCAPE. Intel provided us with an ESCAPE system with two Xeon CPU hosts connected over Intel Ultra Path Interconnect (UPI) links to a Stratix 10 FPGA, with 16 GB of the DDR3 memory on the FPGA were mapped into both hosts address space to allow it to be used as shared-memory. All shared-memory reads and writes were subject to filtering rules configured onto the FPGA. We extended the CLOSURE HAL library to perform memory-mapped copies into the shared memory. This support allowed CLOSURE partitioned binaries to communicate between hosts using this shared memory with a simple read or write into memory-mapped FPGA memory locations.

- In Phase 3 we also extended the HAL library to support communication by sending and receiving files, which allowed cross-domain communication among CLOSURE partitioned binaries using Arcfield's X-ARBITOR CDS. X-ARBITOR can filter these files, whose format is specified in DFDL automatically generated by CLOSURE.

Integration with an Accredited CDS: In addition to integration with the three heterogeneous GAPS CDS hardware, in Phase 3 we integrated CLOSURE with Arcfield/AFRL's X-ARBITOR, an existing software-based CDS accredited by the NCDSMO:

- We developed a demonstration that shows controlled sharing of information among TAK teams using a CLOSURE synchronization application over X-ARBITOR CDS. In the demonstration, each team (enclave) has a per-enclave TAK server, which its local clients continuously update with locations and events. Our C-based synchronization application subscribes to messages of TAK teams and automatically publishes them to other team servers through X-ARBITOR. TAK ordinarily does not possess the ability to share between enclaves; but through CLOSURE we developed a capability for cross-domain TAK operation.
- X-ARBITOR pipelines were provisioned using CLOSURE outputs (DFDL, filters) to set up the required configuration for the cross-domain deployment.
- We had Joint briefing with Arcfield to US Army DEVCOM on the application of CLOSURE toolchain in conjunction with X-ARBITOR software-based CDS.

Performance Benchmarking: We measured CLOSURE throughput and latency both with applications partitioned by the CLOSURE tools and using benchmarking tools:

- The Phase 2 integrated XILINX ZCU102 Embedded Demonstration involved remote control of a camera, video transcoding, filtering of location metadata, and web-based delivery. The video processing portion of the partitioned application is instantiated on an A53 ARM core and the web server portion is instantiated on a MicroBlaze softcore processor on the FPGA, and the two processors (both running Petalinux) are interconnected by MIND DMA pipeline in each direction. The VHDL for the MIND pipeline is auto-generated by the GE MIND toolchain, from DFDL, which in turn is auto-generated by the CLOSURE toolchain for all cross-domain messages for the partitioned application. With the MIND CDS providing real time filtering of both the control messages and video frames, we improved video data rates from the 10 64K frames/second in Phase 2 to 100 64K frames/second in Phase 3 (using improved CLOSURE implementation of the HAL library and faster codecs).
- In Phase 3 we experimented with CLOSURE performance on the Intel ESCAPE GAPS Security Engine evaluation system. The Benchmarking tool can copy data between its local heap memory created using malloc() and one of three types of memory: local heap, local

mapped memory and ESCAPE FPGA mapped memory. The tool measures throughput and latency for each type of transfer for different copy sizes, copy functions, and number of parallel threads. We are able to achieve 150 MB/s to communicate 64 KB frames between two hosts using the ESCAPE shared memory. Although fast, this was two orders of magnitude less than when simply using heap memory, where we able to achieve the memory bandwidth limit. Although conventional wisdom holds mapped memory should be fast and able to reach memory bandwidth limit, we found it was not realized due to known issues, such as slow and unpredictable page fault eviction and Translation Look-aside Buffer (TLB) shoot-downs.

Transition Surrogate Analysis: A transition partner expressed interest in using CLOSURE to partition a use-case for them in order to allow GAPS CDS hardware protect the main control flow. In particular, GAPS would provide the ability to filter information from their GPS subsystem, using semantic validity checks that may be mission-state-dependent:

- In phase 3 we created a surrogate C++ program that emulates key aspects of the transition partner's system. Our approach was to mimic the objective application's control flow without bringing in all the complexity of their infrastructure or any real-time requirements. We focused on the GPS service functionality and followed/adapted the GPS ICD.
- The surrogate GPS service can run in a separate less trusted security enclave and CLOSURE will insert RPCs to communicate between the untrusted GPS subsystem, without violating the real-time time budget. In the rest of the system, local object GPS instances are replaced by "shadow" implementations that invoke RPCs upon access. C++ objects are constructed, destructed, and accessed remotely through RPC calls. CLOSURE generates the DFDL and partitions the surrogate C++ application for execution on a Linux system.

Contributions to DFTVL Working Group: Highlights of our participation in the DFTVL WG Standardization activity, and effort toward working with existing accredited CDS included:

- Development of a proof-of-concept DAGR language and transpiler prototype
- Extraction of <path,offset,len> maps from DFDL to support low-level match and redact operations in hardware such as the ILIP device
- Proof-of-concept DFDL generation from Java classes
- Joint briefing with Arcfield to US Army DEVCOM on the application of CLOSURE toolchain in conjunction with the accredited X-ARBITOR software-based CDS
- Development of a Natural Rule Language to BPF transpiler prototype (for use with programmable networking hardware) by our team member IIT

Additional transition and integration outreach: activities include:

- Invited briefing on CLOSURE toolchain at DARPA V-SPELLS Kick-Off (attached)
- Poster on CLOSURE toolchain in the DARPA GAPS at the ERI 2022 Summit
- Supported GAPS Technical Interchange Meetings focused toward integration and transition including hosting the session on software verification at Peraton Labs.
- Interactions with Arcfield to show CLOSURE capabilities on top of X-ARBITOR (pipeline configuration).
- Multiple telecoms with DARPA, and transition partners to discuss CLOSURE capabilities.
- Development of prototype C++ tools for other DoD use-cases identified by DARPA.

VI. Important Findings and Conclusions

1. Near complete C coverage based on rigorous constraint-model, successfully tested on a several example programs.
2. CLOSURE methodology is general, extends well from C, Java, and C++ and was able to co-exist with other toolchains such as oneAPI, LLVM, JDK, and ADK.
3. Object Oriented languages are complex, there are various constructs that require further investigation before supporting them fully in CLOSURE.
 - a. Existing C++ and Java compilers provide inadequate annotation support for Lambda functions.
4. Focusing on oneAPI/SYCL constructs rather than larger C++ language-base may provide a sweet spot for creating a useful toolchain without addressing the complete C++ language.
5. DFDL/DAGR/DFTVL contributions offer broader applicability to the CDS world.
6. HAL decoupling of device from application was found to be a good design. Enabled rapid support and migration of applications to four heterogeneous GAPS CDSs: including the three GAPS CDSs (based on PCIe bus, DMA and shared-memory), as well as the legacy, accredited X-ARBITOR CDS.
7. Dockerized software delivery greatly eases third-party adopters as evidenced by interactions with academic partners and other TA1/3 GAPS performers.
8. Use of an annotation overlay as opposed to extending the languages offers a number of benefits including: 1) reuse of existing systems i.e., LLVM, 2) reduced implementation effort vs changing a complex compiler such as Clang, 3) gentler learning curve for developers.
 - a. Although changing the language offers more power and control, we believe our approach is more suitable for the needs of the GAPS program (for reasons stated).
9. The state-of-the-art in CDS involves separate development of software for each enclave. The cross-domain sharing policy as well as apportioning of functionality to each enclave is relatively static in today's state of practice. The annotation-driven partitioning of applications with agile redistribution of functionality is considered way ahead of the curve for stakeholders. It may be beneficial to explore other transition domains beyond CDS and MLS, which may be more open to becoming early adopters of CLOSURE technology.
10. Intel ESCAPE TA1 device offers the potential for future high-performance CDS for challenging HPC and AI workloads. We developed approaches that allow us to leverage this device at a proof-of-concept level and understand performance limitations. Realizing the potential of this will require the identification of a compelling cross-domain application use-case and act as a driver for additional GAPS CLOSURE technology development.

VII. Significant Hardware Development

Not applicable.

VIII. Special Comments

None.

IX. Implications for Further Research

Based on our results and findings during CLOSURE, we recommend further research on:

1. We have a relatively complete and usable solution for the C Language that has been tested and demonstrated on multiple non-trivial applications at ERI Summit 2021, End of Phase 1, and End of Phase 2, and Phase 3 use-cases. Our explorations with Java, C++ and oneAPI suggest that developing a complete and usable capability for any one of these complex languages will take several dedicated man-years of effort. We recommend focusing on one (perhaps C++ given its wide applicability) to achieve a more lasting result.
2. Making cross-domain filter transform rules easier to specify and modify by the non-expert and additional support for auto-generation of data format specifications. Further development and maturation of the CLOSURE DFDL auto-generation tools, the DAGR language specification, and a reference DAGR transpiler would result in a capability that is more generally useful to the CDS community. Demonstrations that combine the use of selected CLOSURE tools with an accredited CDS will help gain traction within the CDS community.
3. The CLOSURE toolchain and methodology developed on CLOSURE (basic research / 6.1 effort) provides an excellent foundation to support additional transition program(s). We anticipate that the needs of a complex and realistic DoD mission system will provide emergent research challenges for future efforts.
4. CLOSURE techniques could be moved “left” in the development life-cycle to encompass the system engineering steps of requirements and design. Production CDS applications will undergo significant policy analysis before it gets to the code itself (CLOSURE’s starting point). In order to verify that the intended requirements/design for a policy is properly implemented by the developer in code, GAPS/CLOSURE will need to provide support in these earlier steps of the software waterfall. Future work may consider model-driven design and integrating with tools in this space.
5. Constraint analysis on the partitioned sources (downstream analyzer) requires additional work. The RPCs introduce auto-generated labels (historically needed for Phase 1 verification) that cause conflicts for the downstream analyzer as well as label inconsistencies—unlabeled variables/functions are assigned differently in the upstream and downstream analyses. This approach requires more thought if the same constraint model and formal analysis is to be applied at the upstream and downstream stages.

X. Other Items of Interest

None.

XI. Detailed Technical Documentation

See CLOSURE Toolchain Handbook for C/C++ and Java. Available online (links provided in §I) and attached in this report. CLOSURE publications listed below:

- [1] KSplit: Automating Device Driver Isolation. 2022 USENIX Symposium on Operating Systems Design and Implementation (OSDI). Yongzhe Huang, Vikram Narayanan, David Detweiler, Kaiming Huang, Gang Tan, Trent Jaeger, Anton Burtsev.
- [2] The taming of the stack: Isolating stack data from memory errors. In 2022 Network and Distributed System Security Symposium (NDSS). Kaiming Huang, Yongzhe Huang, Mathias Payer, Zhiyun Qian, Jack Sampson, Gang Tan, and Trent Jaeger.
- [3] N. Sultana. 2021. Leveraging In-Network Application Awareness. In Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration (NAI'21).

Association for Computing Machinery, New York, NY, USA, 63–67.

DOI:<https://doi.org/10.1145/3472727.3472806>

- [4] H. Nagda, R. Nagda, N. Sultana and B. T. Loo, "Demo: Disaggregated Dataplanes," 2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS), 2021, pp. 1095-1097, doi: 10.1109/ICDCS51616.2021.00109.
- [5] M. Levatich, R. Brotzman, B. Flin, T. Chen, R. Krishnan, and S. Edwards, "C-Program Partitioning with Fine-Grained Security Constraints and Post-Partition Verification" , IEEE Military Communications Conference (MILCOM), pp. 285-291, November 2022.
- [6] M. Kaplan, "Study: CLOSURE Integration with Accredited CDS (X-ARBITOR) for DARPA GAPS, January 2024. (available on DARPA Vault)
- [7] M. Kaplan "Surrogate Application Design, Partitioning Concept, and Message Types." Presentation, November 27, 2023. (available on DARPA Vault)
- [8] Huang, Z.; Jaeger, T.; and Tan, G. "Fine-grained Program Partitioning for Security." 14th European Workshop on Systems Security (EuroSec), pages 21–26, 2021
- [9] Narayanan, V.; Huang, Y.; Tan, G.; Jaeger, T.; and Burtsev, A. "Lightweight Kernel Isolation with Virtualization and VM Functions." 16th ACM International Conference on Virtual Execution Environments (VEE), 157–171. 2020
- [10] Nik Sultana, John Sonchack, Hans Giesen, Isaac Pedisich, Zhaoyang Han, Nishanth Shyamkumar, Shivani Burad, André DeHon, and Boon Thau Loo, "Flightplan: Dataplane Disaggregation and Placement for P4 Programs." NSDI'21
- [11] Henry Zhu, Nik Sultana, Boon Thau Loo. "Debugging strongly-compartmentalized distributed systems." APDCM'21.
- [12] André DeHon, Hans Giesen, Nik Sultana, Yuanlong Xiao. "Meta-level issues in Offloading: Scoping, Composition, Development, and their Automation." LATTE'21