https://github.com/gaps-closure

DARPA GAPS Hands-On Workshop at ERI Summit

Agile Cross-Domain Systems Development Using CLOSURE Toolchain

Program Manager: Mr. Walter Weiss
DARPA BAA HR001119S0017 (GAPS-TA2)
8/20/2020

Co-PI: Mr. Michael Kaplan
Scientific Research/Analysis Manager
mkaplan@perspectalabs.com

Co-PI: Dr. Rajesh Krishnan
Senior Research Scientist
rkrishnan@perspectalabs.com

perspecta LABS

This material is based upon work supported by the Defense Advanced Research Projects Agency (under Contract No HR001119C0106 The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U S government. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited)

Agenda for Today's Workshop

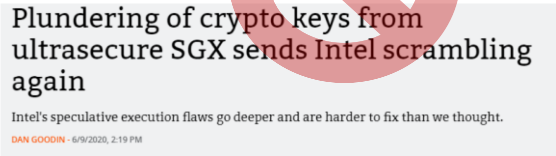
- **Overview Briefing (15 Minutes)**
  - Background on GAPS Effort
  - Overview of CLOSURE technology, tools, and methodology
- **CLOSURE Quick Start (15 Minutes)**
  - How to get started with CLOSURE co-design tools within the XtremeLabs environment
  - Review of CLOSURE Language Extensions (CLE)
- **Instructor-Led Exercises (30 Minutes)**
  - *Exercise 1*: Utilize CLE to express security intent on a simple C program, partition, compile and execute in emulated environment (code and security intent provided).
  - *Exercise 2*: Imagine cross-domain developer/auditor needs to change the policy. Developer must update/replace CLE from exercise 1 and address any refactoring required to satisfy policy.
- **Demonstration on larger application with open-source libraries (10 Minutes)**
- **Independent participant exercise with instructor assistance (50 Minutes)**
  - *Exercise 3: Rework the example program with a third set of policies, compile and test in emulator*

Upon completing this course you will:

- Understand GAPS cross-domain systems development using CLOSURE toolchain
- Gain expertise using emerging (phase 1) GAPS technologies

DARPA MTO Guaranteed Architecture for Physical Security (GAPS)

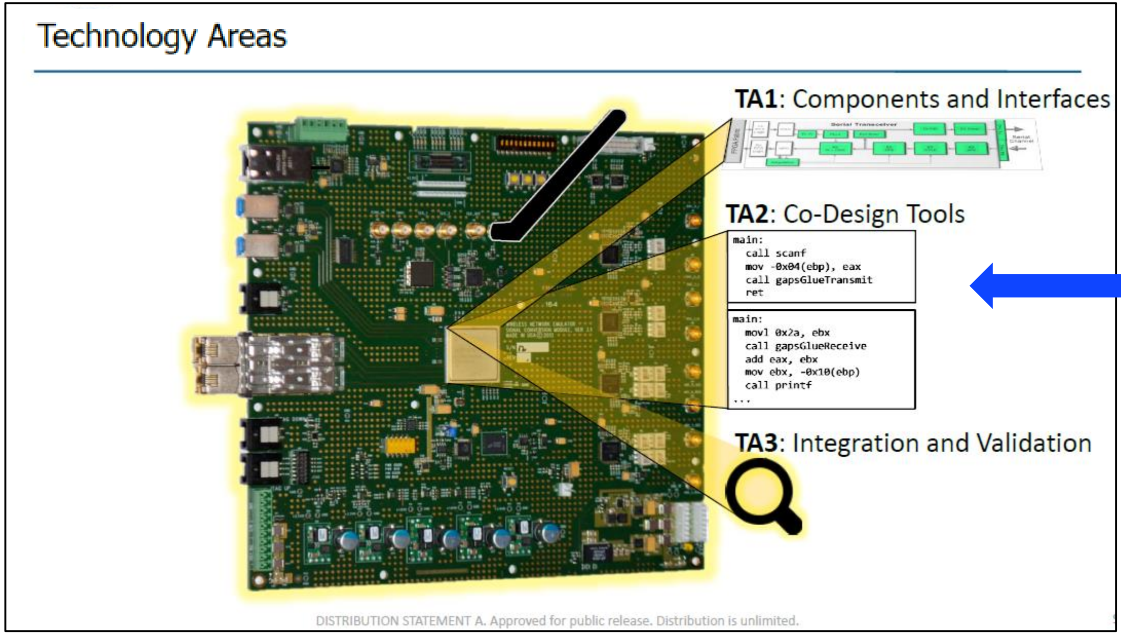
Develop hardware and software architectures with provable security interfaces to physically isolate high risk transactions


Plundering of crypto keys from ultrasecure SGX sends Intel scrambling again

Intel's speculative execution flaws go deeper and are harder to fix than we thought.

DAN GOODIN - 6/9/2020, 2:19 PM

Source: https://arstechnica.com/information-technology/2020/06/new-exploits-plunder-crypto-keys-and-more-from-intels-ultrasecure-sgx/



Source: DARPA GAPS Proposers Day briefing

Perspecta Labs CLOSURE

Started September 2019

Currently 10 months into Phase 1 (Total 3 phases over 4.5years)

GE Research, Mercury Systems, Galois, Perspecta Labs, Northrop Grumman, General Dynamics, and Intel on the program to realize the GAPS vision

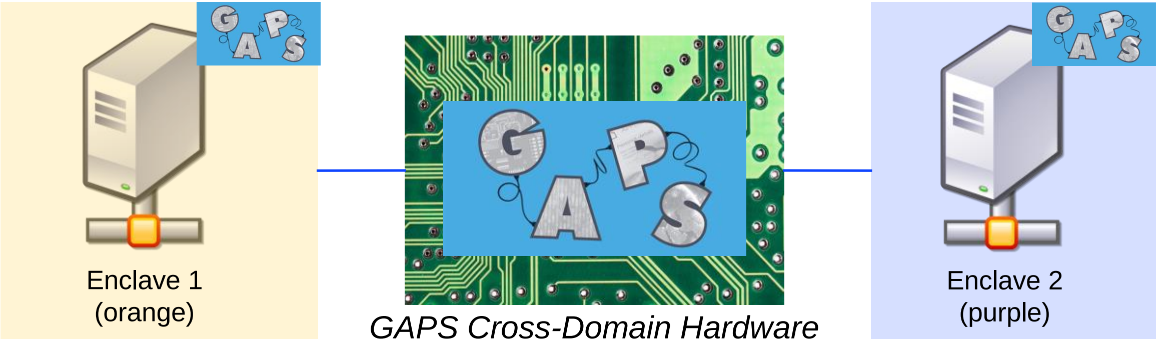
Program Partitioning to Guarantee Physical Isolation of Cross-Domain Transactions

```
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



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

perspecta LABS

CLOSURE Co-Design Workflow

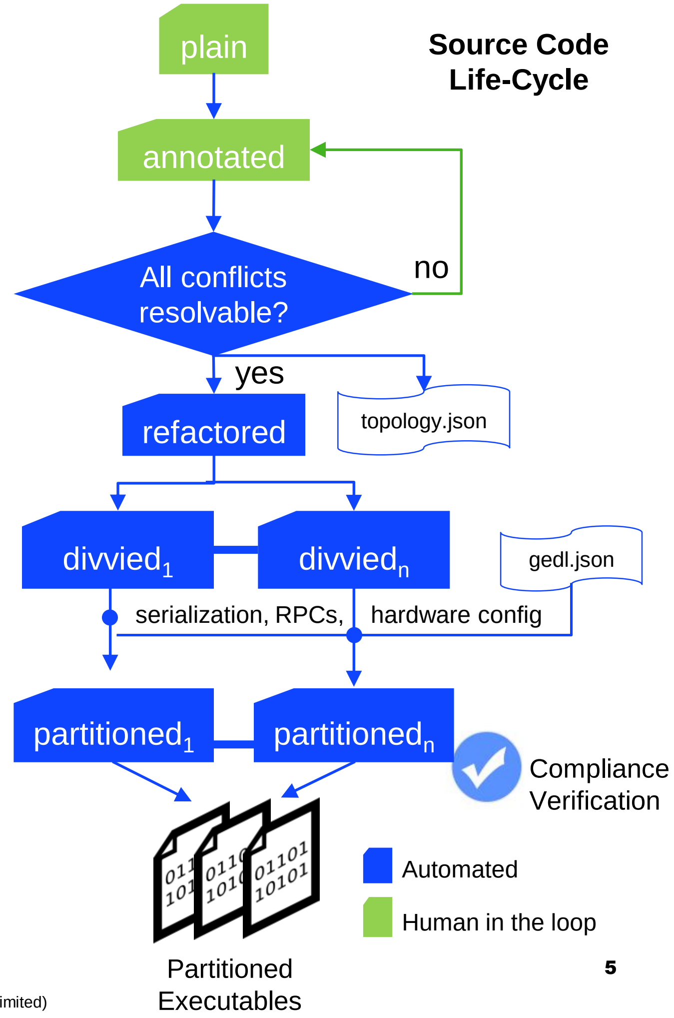
0. Import/Create Cross-Domain program (plain source)

1. Annotate code to express security intent

2. Conflict Analysis for partitioning feasibility

Most developer time spent here

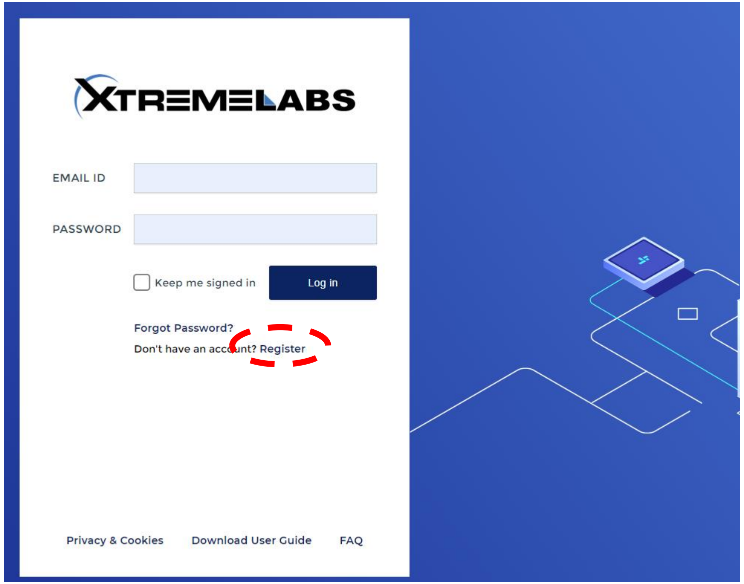
3. Automated Code Generation, Verification, Build, and Test
   - Divides code into per-enclave source trees
   - Automates program rewriting and code generation
     - Serialization, marshalling, remote procedure calls (RPCs), Data Format Description Language (DFDL) spec, Cross-Domain hardware configurations
   - Compiles to LLVM Intermediate Representation for program analysis and verification
   - Runs end-to-end test in CLOSURE emulator



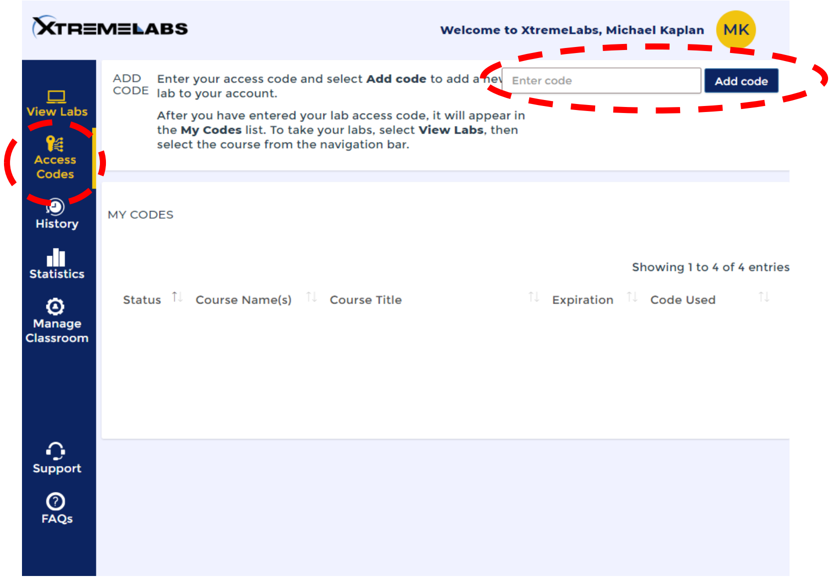
perspecta LABS

Distribution Statement "A" (Approved for Public Release, Distribution Unlimited)

Entering the Lab (1/2)



Source: XtremeLabs



Source: XtremeLabs

1. Register and log in at: https://labs.xtremelabs.io

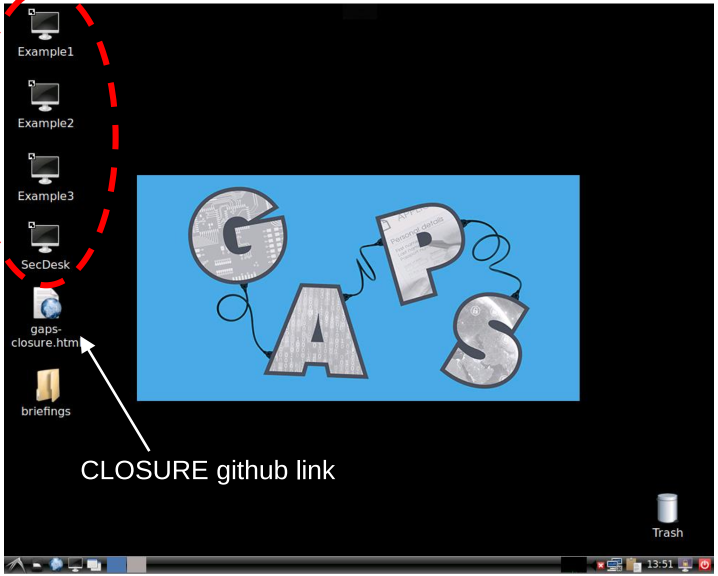
2. Choose “Access Codes” on toolbar. Enter provided code

Entering the Lab (2/2)

Source: *XtremeLabs*

3. Choose "View Labs" on toolbar and click "Take Lab" to launch your lab VM

Shortcuts to examples



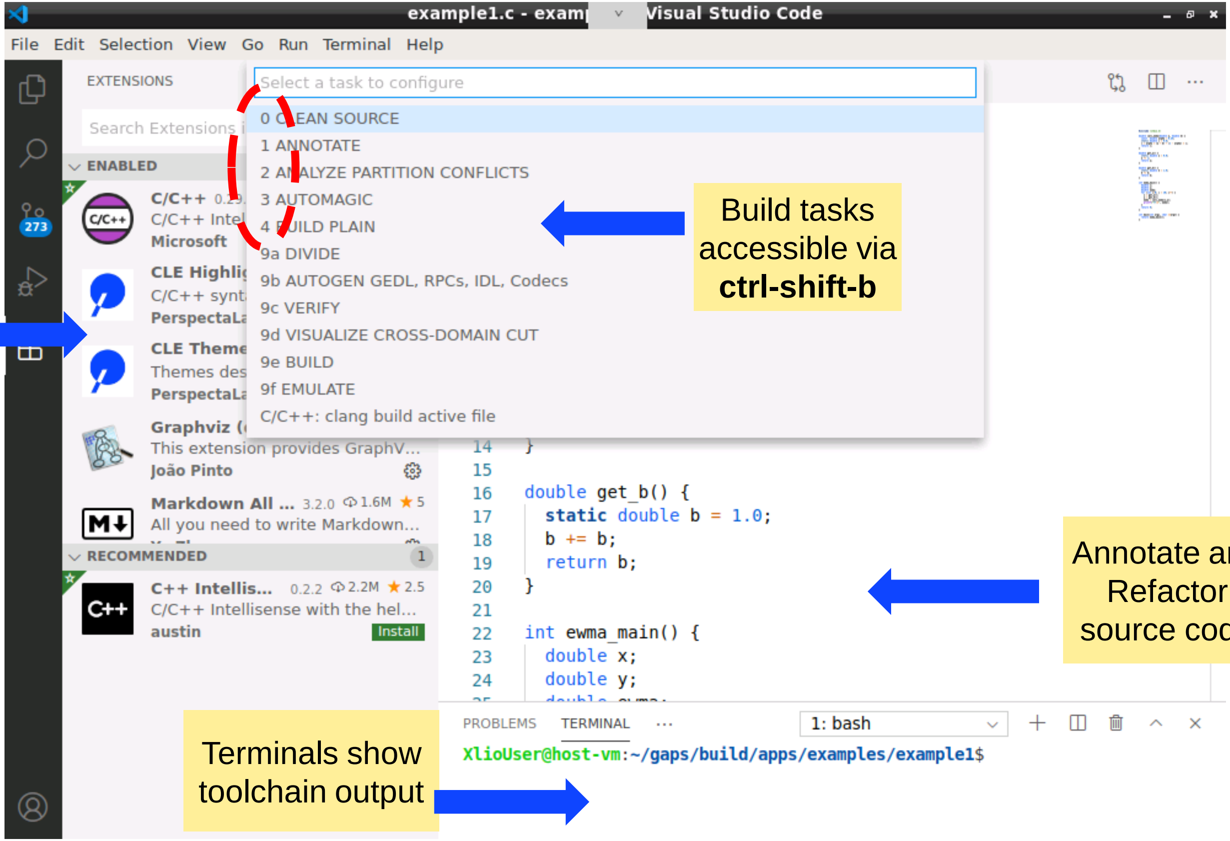
Source: XtremeLabs, DARPA GAPS
https://www.darpa.mil/news-events/guaranteed-architecture-for-physical-security-proposers-day

4. GAPS VM accessible in browser for lab exercises (clicking desktop shortcuts opens exercises in CLOSURE Development Environment)

perspecta LABS

Navigating the CLOSURE Visual Interface (CVI)

CLOSURE Plug-Ins Installed



Build tasks accessible via **ctrl-shift-b**

Annotate and Refactor source code

Terminals show toolchain output

Source: VSCode

Today’s Example Program

- Program consists of functions **`get_a`** and **`get_b`** which return static values **`a`** and **`b`**. Function **`ewma_main`** calls **`get_a`** and **`get_b`** and passes these values to **`calc_ewma`** for a computation. The result is returned to **`ewma_main`** and printed to the screen.
- Original program was written without cross-domain security concerns. We will see how we can use CLOSURE tools to refactor the program to meet different cross-domain security intents.

Exercise 1 Partitioning Intent

- Variable **`a`** in **`get_a()`** is in **`ORANGE`** and can be shared with **`PURPLE`**
- Variable **`b`** in **`get_b()`** is in **`PURPLE`** and cannot be shared
- Calculated **`ewma`** must be available on **`PURPLE`** side (for printing)

Exercise 2 Partitioning Intent

- Variable **`a`** in **`get_a()`** is in **`ORANGE`** and can be shared with **`PURPLE`**
- Variable **`b`** in **`get_b()`** is in **`PURPLE`** and cannot be shared
- Calculated **`ewma`** must be available on **`ORANGE`** side (for printing)

```c
#include <stdio.h>

double calc_ewma(double a, double b) {
  const  double alpha = 0.25;
  static double c = 0.0;
  c = alpha * (a + b) + (1 - alpha) * c;
  return c;
}

double get_a() {
  static double a = 0.0;
  a += 1;
  return a;
}

double get_b() {
  static double b = 1.0;
  b += b;
  return b;
}

int ewma_main() {
  double x;
  double y;
  double ewma;
  for (int i=0; i < 10; i++) {
    x = get_a();
    y = get_b();
    ewma = calc_ewma(x,y);
    printf("%f\n", ewma);
  }
  return 0;
}

int main(int argc, char **argv) {
  return ewma_main();
}
```

CLE Concepts

```
// Precise readings cannot be shared
# pragma cle begin ORANGE
double precise_readings[128];
# pragma cle end ORANGE

// Return cannot be shared, via inference
double kth_reading(int k) {
  return precise_readings[k];
}

// Average can be shared, but human must check
// that only average is shared by this function

#pragma cle begin XDLINKAGE_AVERAGE
double average(double reads[]) {
#pragma cle end XDLINKAGE_AVERAGE
   double ret = 0.0;
   for (int i=0;  i<128; i++) ret += reads[i];
   return ret / 128;
}
```

Annotated C source using CLE

- label
  - level
  - cdf : level → remotelevel
    - guarddirective
    - argtaints, codtaints, rettaints

CLE Schema

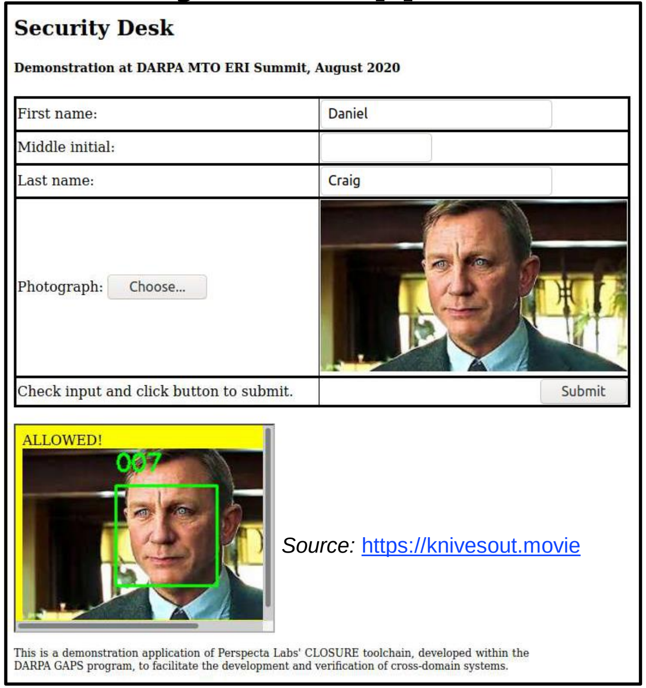
```
#pragma cle def ORANGE {"level": "orange"}
#pragma cle def ORANGE_SHARE { \
  "level":"orange” \
  "cdf": [\
    {"remotelevel": "purple", \
     "direction": “egress", \
     "guarddirective": {"operation": "allow"}
    }]}
#pragma cle def XDLINKAGE_AVERAGE
 {"level": "orange",\
  "cdf": [\
    {"remotelevel": "purple", \
     "direction": "bidirectional", \
     "guarddirective": {"operation": "allow"}, \
     "argtaints": [["ORANGE"]], \
     "codtaints": [], \
     "rettaints": ["ORANGE_SHARE"] \
    }]}
```

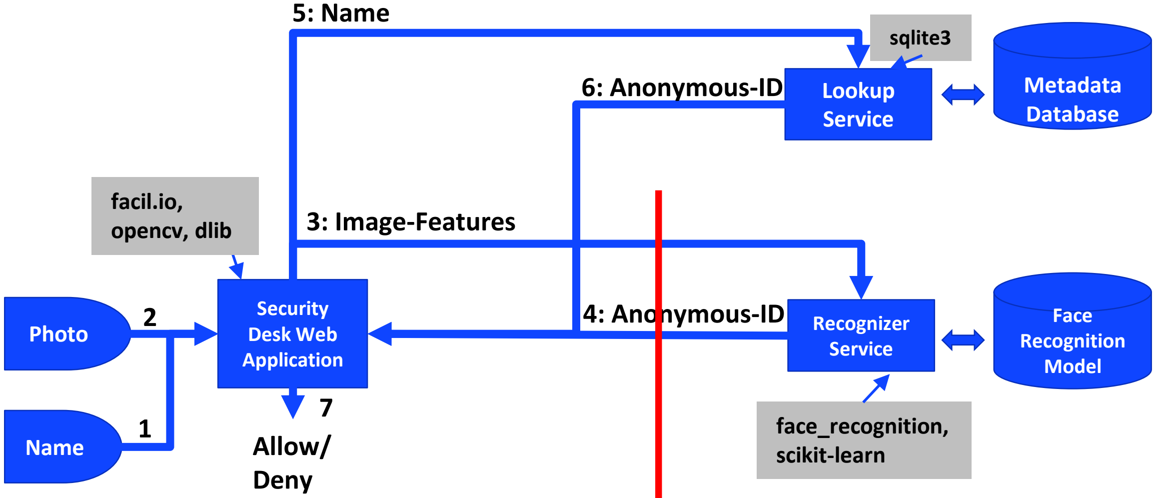
CLE Definitions

perspecta LABS

Instructor-Led CLOSURE Walkthrough Session...

Security Desk Application with Face Recognition





1,2,7: Web form via browser
- Request: POST name, image
- Response: allow/deny, recognizer overlay

Cross-Domain Partitioning Intent: Isolate face recognition from rest of system

Open Source Technologies:
- facil.io: web application framework (C, embedded)
- sqlite3: database (C API, library)
- face_recognition: opencv, dlib, scikit-learn (Python/C API, python3, and C/C++)

SLOC	Directory	SLOC-by-Language (Sorted)
25576	facilio	ansic=25576
478	top_dir	ansic=439,python=39

Exercise 3

Based on the following security intent and objectives, annotate the program on the right such that:

- Variable a in get_a() is in ORANGE and cannot be shared
- Variable b in get_b() is in ORANGE and cannot be shared
- EWMA must therefore be computed on ORANGE; EWMA is sharable to PURPLE. Calculated EWMA must be available on PURPLE side (for printing)

We encourage you to try this on your own!

We are here to help with questions and can interact with your VM.

Use the Zoom chat window to ask questions. You can also raise your hand, and when we recognize you, unmute to talk.

Thank you

For additional information, please contact GAPS@darpa.mil

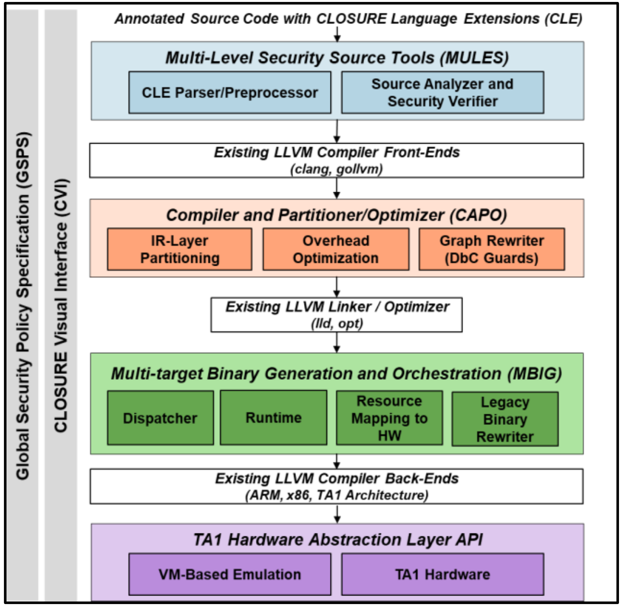
perspecta LABS

Additional Background

Overview of CLOSURE

CLOSURE innovations address key challenges of GAPS TA2

- Language extensions for multiple languages (focus on C/C++ and Golang) for security annotations
- Automated pointer-aware program partitioning
- Parametric optimization of program partitioning
- Program rewriting to insert IPC and guards using a Design-by-Contract methodology
- Constraint-solver based mapping of software to hardware elements drives target-specific binary generation & optimization
- An emulation capability for development, testing, performance evaluation, and verification of the partitioned solution
- A visual interface for editing, debugging, visualizing source and intermediate forms, managing the development and optimization workflow, and emulation-based testing



GAPS Technical Challenges Addressed by CLOSURE

How to characterize and express cross-domain security intent in software?

Annotation Language Specification

How to analyze the annotated source to identify conflicts, and guide the developer towards compliant programs?

Security Conflict Analysis and Guided Refactoring

How to verify the partitioned programs and formally argue their correctness and security compliance?

Program Verification

How to automate portions of cross-domain partitioning and refactoring of programs?

Cross-Domain Code Generation and Guard Configuration

How to optimize cross-domain programs to reduce overhead and map functionality to target hardware topology?

Integer Programming over Partitioning Constraints

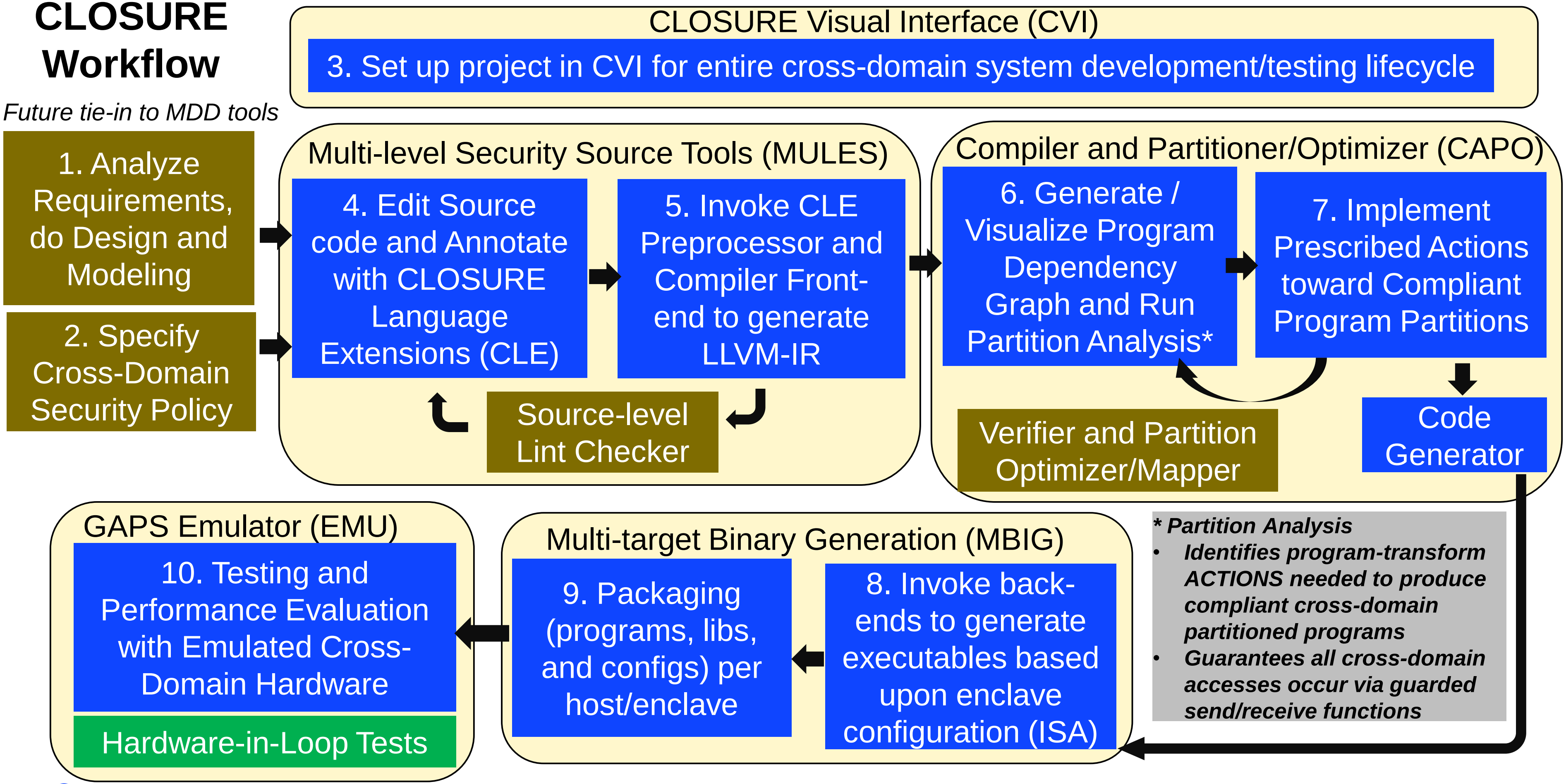
How to create an ecosystem for developers and foster GAPS technology adoption?

Tools, Examples, Standards, and Documentation

perspecta LABS

CLOSURE Workflow

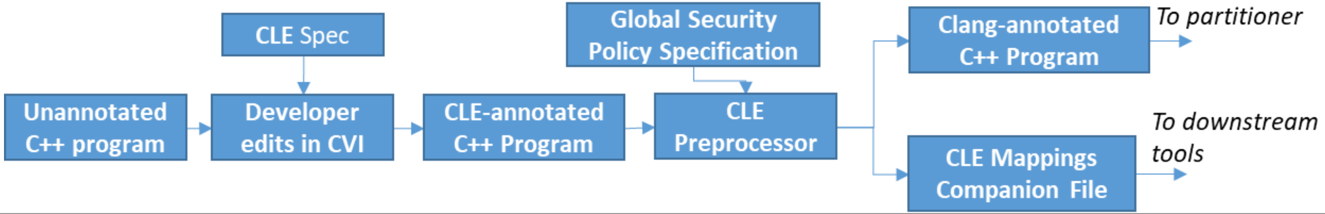
Future tie-in to MDD tools



Needs GAPS Hardware | Demonstrated Capability | Future Tooling

CLOSURE Language Extensions (CLE): Preprocessor and Security Annotations for Variables

- Designed to be toolchain agnostic
- Current draft available on github
- Language extensions being standardized by GAPS community



Developer defines CLE labels and associated security policies

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

Developer annotates code with CLE label

```
#pragma cle begin ORANGE
GpsSensor* gps = new GpsSensor(p, v);
#pragma cle end ORANGE
```

CLOSURE language extensions enable developers to intuitively express cross-domain security concerns using annotations within application source code. CLOSURE co-design tools, driven by the annotations, lead to rapid development and deployment of cross-domain applications that are correct-by-construction.

A ***label*** defines a security type. Associated with the ***label*** is a ***level*** plus constraints on cross-domain data sharing (**cdf**). All data marked with a given ***level*** must reside in one enclave. Data in one enclave may have different labels – some may not be shared while others may be shared, possibly after redaction. The ***guarddirective*** within ***cdf*** associated with the ***label*** specifies such data sharing constraints.

CLOSURE Language Extensions (CLE): Security Annotations for Functions

Developer specifies approved CLE label taints for each portion of function

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

Developer annotates function declaration with CLE label

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

In addition to indicating which partition to place the function, function annotations specify the developer-approved CLE labels for the input arguments (`argtaints`), code body (`codtaints`), and return value (`rettaints`).

The taints indicate to the **conflict analyzer** the developer's intent with regard to mixing data of different labels (but same level).

All functions called by CLOSURE cross-domain RPCs must be "***blessed***" with an annotation, otherwise conflict analyzer will reject.

perspecta LABS

Exercise 1

Based on the following security intent and objectives, annotate the program on the right such that:

- Variable a in get_a() is in ORANGE and can be shared with PURPLE
- Variable b in get_b() is in PURPLE and cannot be shared
- Calculated EWMA must be available on PURPLE side (for printing there)

(CLE label definitions, JSON formatted, placed at top of source file)

```
#pragma cle def PURPLE {"level":"purple"}

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

```
#include <stdio.h>
double calc_ewma(double a, double b) {
  const  double alpha = 0.25;
  static double c = 0.0;
  c = alpha * (a + b) + (1 - alpha) * c;
  return c;
}

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 PURPLE
  double ewma;
  #pragma cle end PURPLE
  for (int i=0; i < 10; i++) {
    x = get_a();
    y = get_b();
    ewma = calc_ewma(x,y);
    printf("%f\n", ewma);
  }
  return 0;
}

int main(int argc, char **argv) {
  return ewma_main();
}
```

Call to get_a will be wrapped with CLOSURE RPC to securely marshal 'a' from ORANGE to PURPLE

Exercise 2

Based on the following security intent and objectives, annotate the program on the right such that:

- Variable a in get_a() is in ORANGE and can be shared with PURPLE
- Variable b in get_b() is in PURPLE and cannot be shared
- Calculated EWMA must be available on ORANGE side (for printing there)

(CLE label definitions, JSON formatted, placed at top of source file)

```
#pragma cle def PURPLE {"level":"purple"}

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

```
#include <stdio.h>
double calc_ewma(double a, double b) {
  const  double alpha = 0.25;
  static double c = 0.0;
  c = alpha * (a + b) + (1 - alpha) * c;
  return c;
}

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;
}

int main(int argc, char **argv) {
  return ewma_main();
}
```

Unresolvable Error: ‘b’ in ‘get_b’ cannot be shared from PURPLE to ORANGE. ewma_main must be refactored to satisfy security constraints