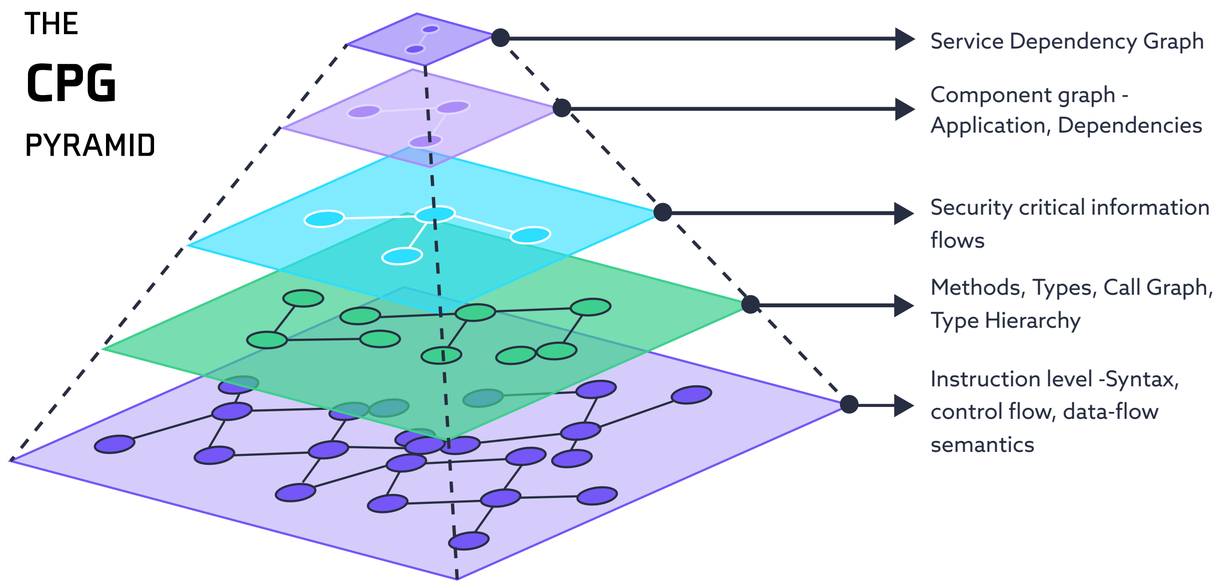
ShiftLeft

Code Property Graph

What is the code property graph?


THE
CPG
PYRAMID
Service Dependency Graph
Component graph - Application, Dependencies
Security critical information flows
Methods, Types, Call Graph, Type Hierarchy
Instruction level -Syntax, control flow, data-flow semantics

CPG constitutes the code represented in the form of a layered graph that can be queried to obtain security relevant information about the code. For each statement/expression of the code - whether in bytecode form or source code form - a syntax tree is included in the code property graph that decomposes it into its language elements. These syntax trees are connected via control flow edges to form control flow graphs. Taking into account known semantics of library functions, a data flow representation is constructed that enables interprocedural data flow analysis similar to that possible via system dependence graphs. Throughout the analysis, additional nodes and edges are created that represent analysis results, however, without introducing additional external information. The complete representation is packaged into a binary that is stored in encrypted cloud storage and cannot be accessed or decrypted.

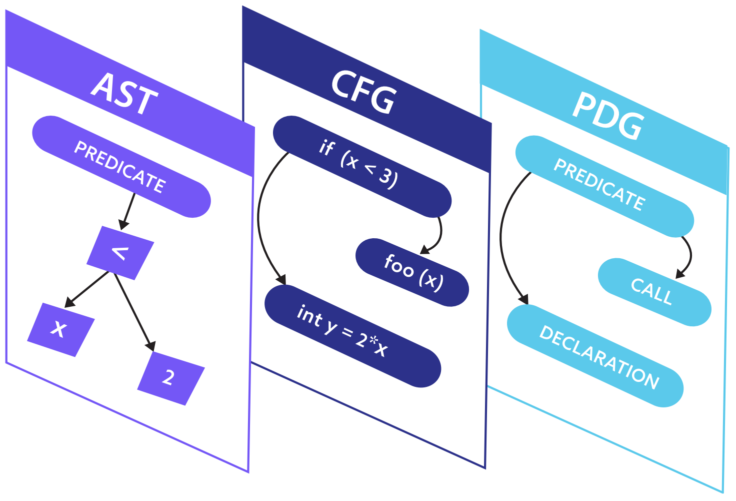
Is the code property graph specification open source?

Yes, the individual elements constituting the CPG and their specifications are outlined below:

1. https://docs.shiftleft.io/core-concepts/code-property-graph
2. https://github.com/ShiftLeftSecurity/codepropertygraph

How is code mapped to this intermediate representation?

```
...
if (x < 2)
    foo(x);
else
    int y = 2*x;
...
```



For Java, code property graphs are created from Java Archives (JARs), and hence, from the bytecode representation of the program. In contrast, for Javascript, C, C++, C#, Golang and Python, they are constructed from source code. Configuration files are analyzed selectively for secrets detection but are largely ignored to date. Dependencies are named in the code property graph, but their implementations are skipped whenever such implementations are detected. The build configuration is not included in the code property graph to date.

Example : https://docs.shiftleft.io/ngsast/analyzing-applications/java

What is the ondisk-based representation of a CPG?

Code property graphs as created by the `sl` tool are serialized property graphs, that is, they are given by a set of nodes carrying key-value pairs and a set of labeled edges that connect these nodes, where edges may also carry key-value pairs. Depending on the source programming language, the graphs are stored either as ShiftLeft OverflowDB graph database files or as Google protocol buffers. Both of these are optimal binary data representations as opposed to textual representations such as JSON or XML.

What is the constitution of a CPG and how does it differ from source code?

Source Code (depending on the programming language) consists of variables, literal, expressions, functional blocks, scoped modules/packages, macros and type information. Depending on the LR grammar of the programming language, all of these properties and represented on a contextual graph

The basic idea is to parse the source code, represent it as a graph which extends on the standard abstract syntax tree (AST) to include information about data and control flow, and store it in a graph representation.

Besides the AST, the CPG combines:

- the control flow graph (CFG), which represents the order in which statements are executed depending on the conditions, and
- The program dependence graph (PDG), which tells us how taint (attacker controlled or exposed variables) travel to security sensitive functions after being validated, transformed, etc.

CPG is also programming language agnostic.

In summary CPG contains the AST + CFG + PDG. No configuration, environmental setup scripts, etc are carried over into the graph.

What are the current languages supported and how long does it take to onboard a new language?

ShiftLeft currently supports : **Java, .NET (C#), JSP, JavaScript, TypeScript, Python, Golang, C, C++, Scala, Kotlin (Coming Soon!)**.

Depending on the type of programming language, strongly typed takes approximately ~2-3 months and untyped or typed inferenced (dynamic) takes approximately ~4-8 months. The basis/foundation was set when we onboarded Java (strongly typed) and JavaScript (untyped).

How and where can the CPG be created and what are the security controls established in the SaaS infrastructure?

The CPG can be created in the customer's CI sandbox using ShiftLeft's ***sl*** command

```
sl analyze --app <name> --java [<path-to-JAR/WAR>]
```

This command can be executed as a CI action in the CI sandbox (Jenkins/Travis/Circle/GitHub Action/Azure pipelines, etc) leading to the CPG being created (and obfuscated) on the sandbox. Thereafter the graph is compressed, signed and securely transmitted to the tenant account in our SaaS cloud.

The on-disk representation of the CPG is encrypted using server-side encryption with customer master keys (CMKs) stored in a Key Management Service.

The encrypted format on SaaS storage units are only accessible by our internal APIs. Even upon access, they can only be parsed by ShiftLeft's proprietary tools and hence, on-disk the graphs are extremely secure. All access points are restricted based on Admin RBAC controls governed by SOC-2 type 2 standards.

Can the CPG be reverse engineered or reconstituted to the target source representation?

No, it cannot be reversed for the the following reasons

1. The representation (based on CPG specification) contains only certain properties (metadata representing syntax trees, variable declarations, blocks, scope, type information, etc) and NOT the full code representation
2. The representation (as CPG) is in a binary/obfuscated form that cannot be queried upon or loaded without ShiftLeft's proprietary toolchain (for analysis)
3. All CPGs stored in ShifLeft's SaaS infrastructure and governed and restricted by automated controls defined by SOC-2