Security Signals

A framework to scale web security

Sławomir Goryczka
Information Security & Software Engineer
Google Switzerland
✉ *slawek@google.com*

Michele Spagnuolo
Staff Software Engineer
Google Switzerland
✉ *mikispag@google.com*

Agenda

01 Introduction to Web Security

02 Collecting Signals

03 Processing Signals

04 Using Data to Improve Security

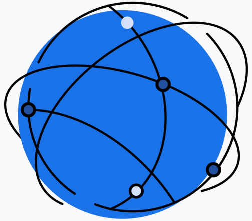
05 Use Cases

06 Example: Cross-Site Request Forgery

01

Introduction to Web Security

Web Security



Web services accept HTTP requests from users and return HTTP responses with relevant data.

- **Attack surface** corresponds to the set of actions that can be invoked directly or via client browser on the target service.
- **Vulnerabilities** can generally be triggered by sending HTTP requests.
- **HTTP Headers** on the HTTP request/response level are often sufficient to gain an understanding of both potential attacks and applied defenses or mitigations.

Why is Web Security hard, especially at Google?

Possibly the largest number of web application in the world:

- more than 8000 web services,
- services are hosted across almost 1000 registrable domains,
- processing trillions of HTTP requests from billions of web users daily,

... serving web pages created and persisted by a heterogeneous ecosystem with:

- many programing languages, e.g. Java, C++, Python, Go,
- HTML template system engines, sanitizers,
- Billions of line of code, thousands of third-party libraries,

... changing all the time.

Secure-by-Design or Fail to Scale

With a large-scale, rapidly evolving codebase, fixing vulnerabilities one-by-one is neither efficient nor scalable.
To make security an ambient property of the developer infrastructure, the following is needed:

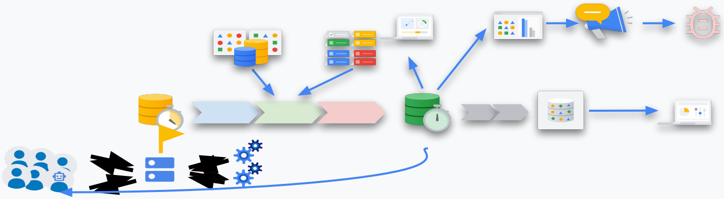
- **Guidelines and recommendations** for developers,
- **Tools, libraries,** and **frameworks**,
- A **"well-lit path"**,
- **Security evaluation and justification of non-recommended approaches**,
- **Fixing regressions**, blind spots, etc.

Security Signals Framework

Security Signals is a framework to collect static and security-related usage data (aka signals) about a web ecosystem to generate insights, report bugs, or prioritize work.
It can also provide higher-level interpretations of the data to:

- Provide **visibility** into security stance of the web infrastructure,
- **Optimize resource allocation**, by evaluating web application risk,
- Determine if certain applications are inherently "**secure-by-design**" from broad classes of vulnerabilities,
- Provide **continuous monitoring** of security controls and assurance of the alignment to the "secure-by-design" principles.

Security Signals Components



The main component is a scalable **collection mechanism** of runtime security signals, which is:

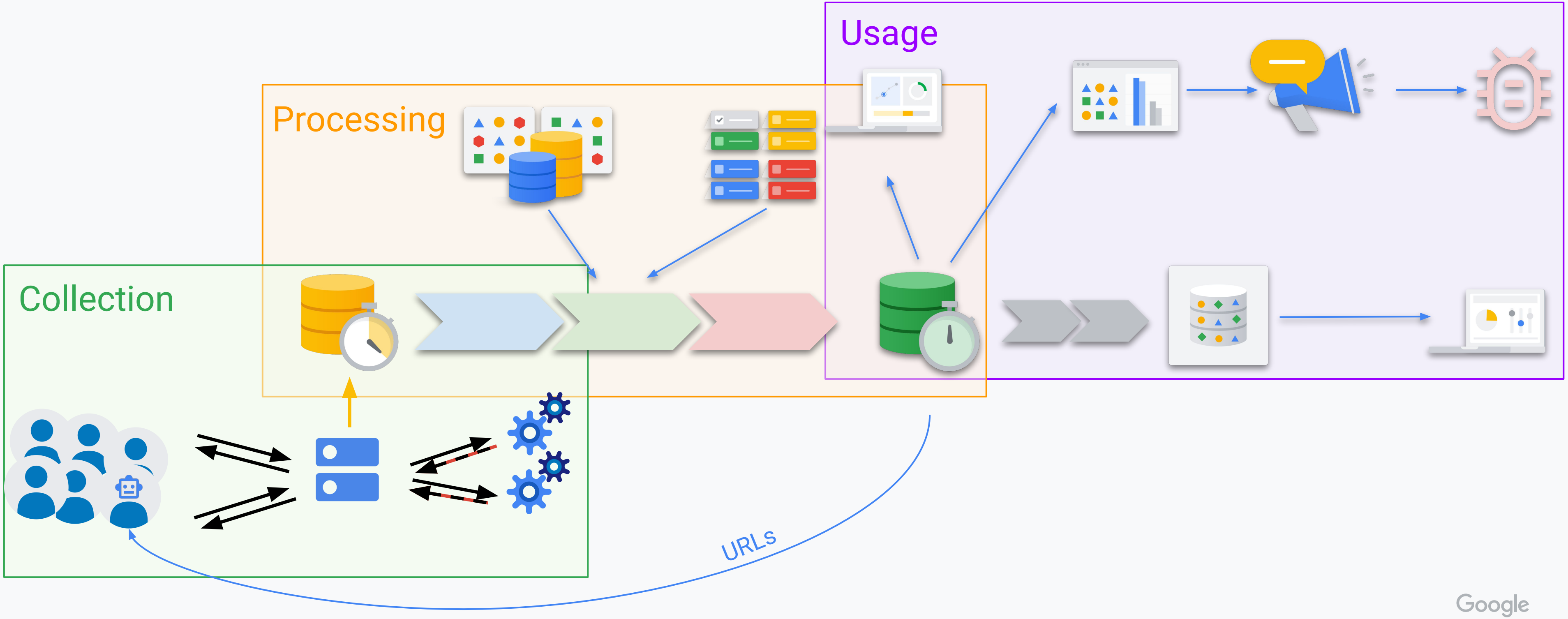
- **Technology-agnostic,**
- **Comprehensive**.

… and a **batch map-reduce pipeline**, which joins signals with auxiliary data and generates security-relevant insights.

Additional tools:

- **Alerting** when detecting anomalies or regressions,
- **Automated bug reporting** and assignment.

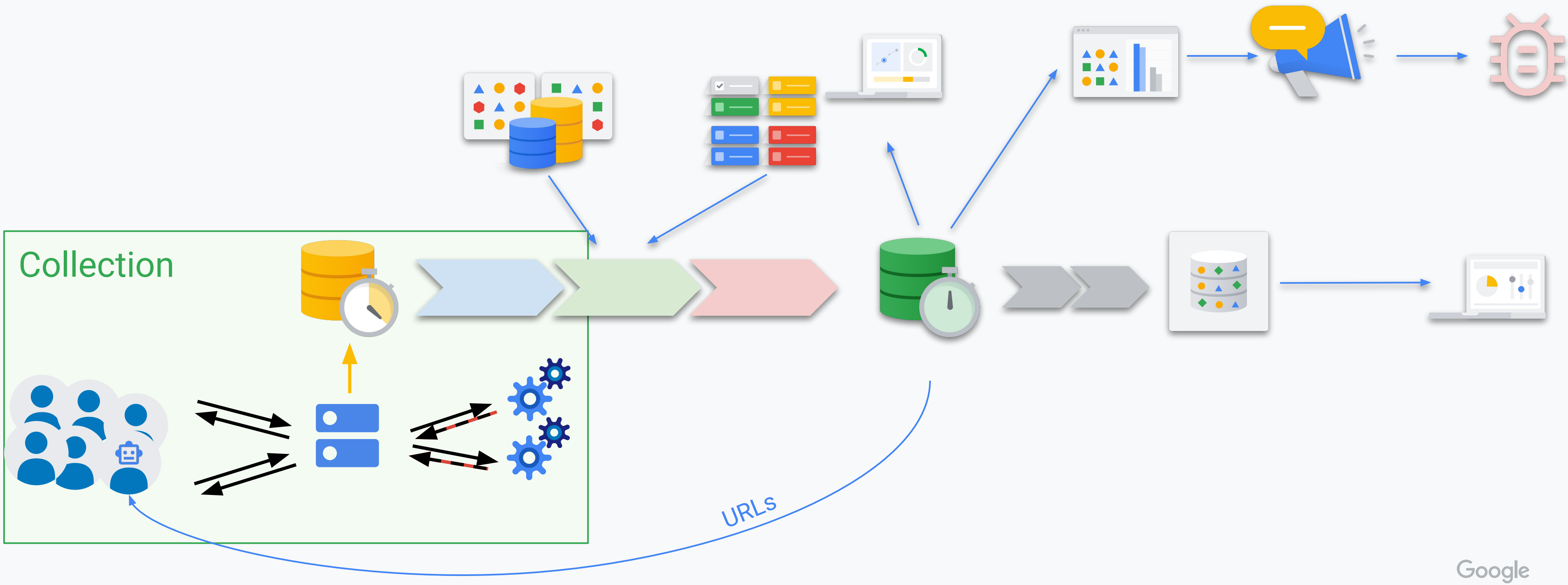
Security Signals Architecture



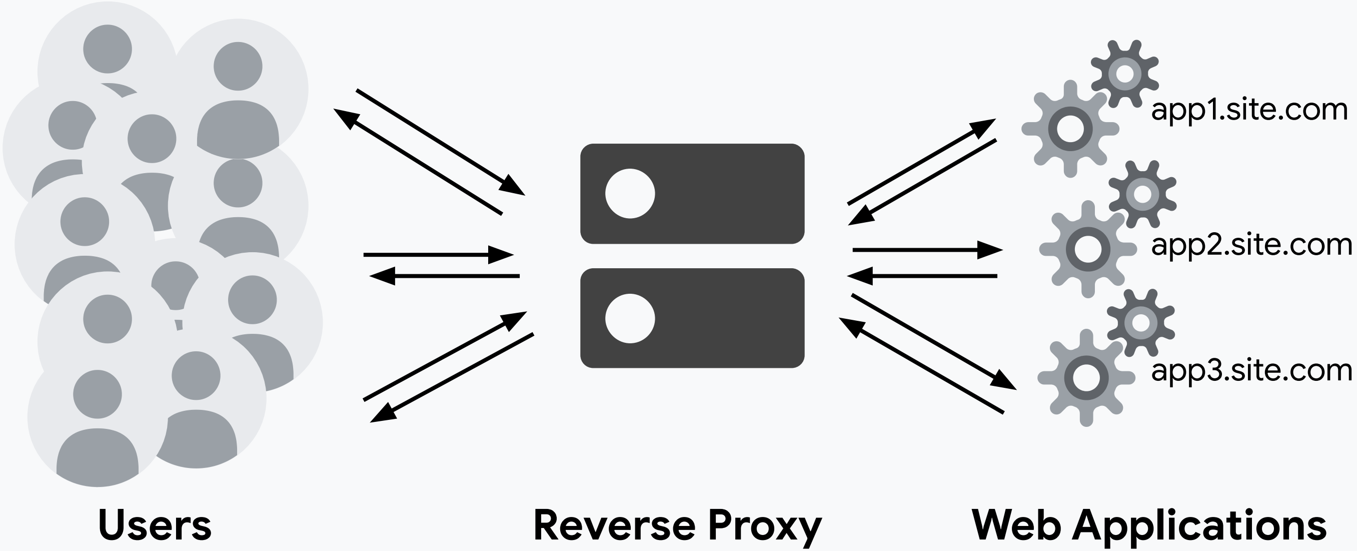
02

Collecting Signals

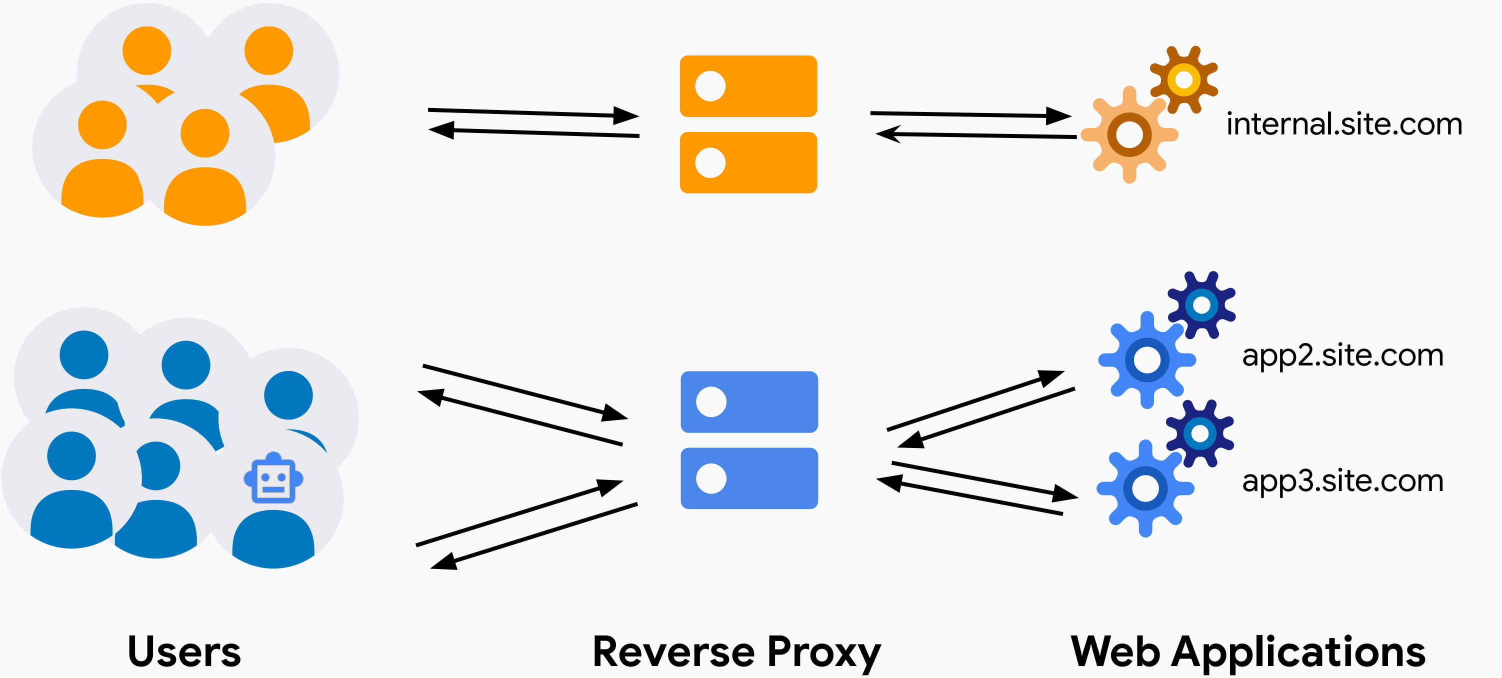
Security Signals Architecture



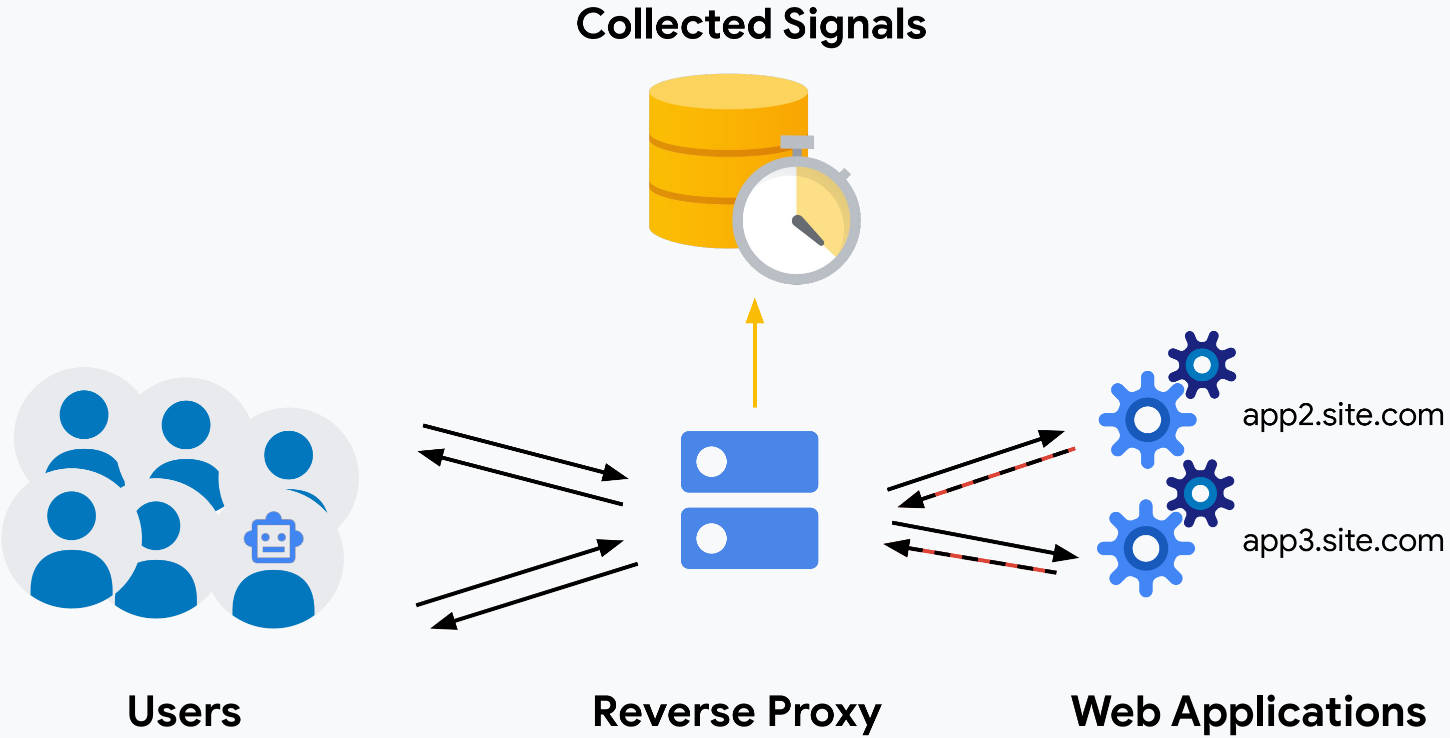
Web Traffic Flowing Through a Reverse Proxy



Web Traffic Flowing Through a Reverse Proxy



Collecting Security Signals



Collecting Data: Challenges

Google processes trillions of HTTP requests from billions of web users daily. To ensure privacy of users, feasibility and quality of generated insights:

- **Web traffic is sampled** with a rate of usually up to 1%, and 10% for internal traffic,
- **Sensitive data** and request/response **bodies are not collected**,
- Individual HTTP requests/responses are not persisted for a long time – **only aggregated and de-identified data**,
- A very short **retention time**,
- **Isolation** of persistent data with **audited access**, and only **justified human access**,
- **Stability** and **functionality** of the GFE.

Collecting Basic HTTP Request & Response Data

- HTTP method,
- Destination host,
- Redacted path and no query parameters!
- Status code,
- Returned MIME type,
- `Referrer-Policy`,
- `Cache-Control`,
- User agent: only browser name and the main version,
- Cookies: security attributes, no value!

Nothing about and from the HTTP request/response body is collected.

Collecting Security-Related HTTP Headers

Web platform security mechanisms are generally configured through HTTP response headers; similarly, clients often provide security-related information in request headers, which Security Signals collect:

- `Content-Security-Policy`,
- `Cross-Origin-Embedder-Policy`,
- `Cross-Origin-Opener-Policy`,
- `Cross-Origin-Resource-Policy`,
- `Sec-Fetch-*`,
- `Strict-Transport-Security`,
- `X-Content-Type-Options`,
- `X-Frame-Options`,
- …

Synthetic Security Signals

Synthetic signals are a core capability of the Security Signals approach. They contain additional metadata that is not normally included in the HTTP response.

They are:

- Generated by instrumented web frameworks,
- Using an internal-only `X-Google-Security-Signals` HTTP response header,
- Collected when passing reverse proxy…
- … and dropped before sending outside.

Collected Synthetic Security Signals

Request-scoped synthetic signals (examples):

- `TEMPLATE`: The server-side templating system that generates HTML output.
- `CSRF`: The presence of Cross-Site Request Forgery protections to verify if an CSRF check was carried out by the backend on state changing requests.
- `SEC_FETCH`: The presence of server-side isolation policies to assess if isolation policies were applied to prevent cross-site attacks.

Service-scoped synthetic signals:

- `FRAMEWORK`: The serving web framework.
- `ACTION`: A pointer to the method/function generating the web response.
- `BUILD`: Information about the application’s build environment.

Auxiliary Data and Risk Signals

Auxiliary data are collected from internal databases. They enrich security signals with information about:

- the production environment,
- product and ownership information,
- source-code information, etc.

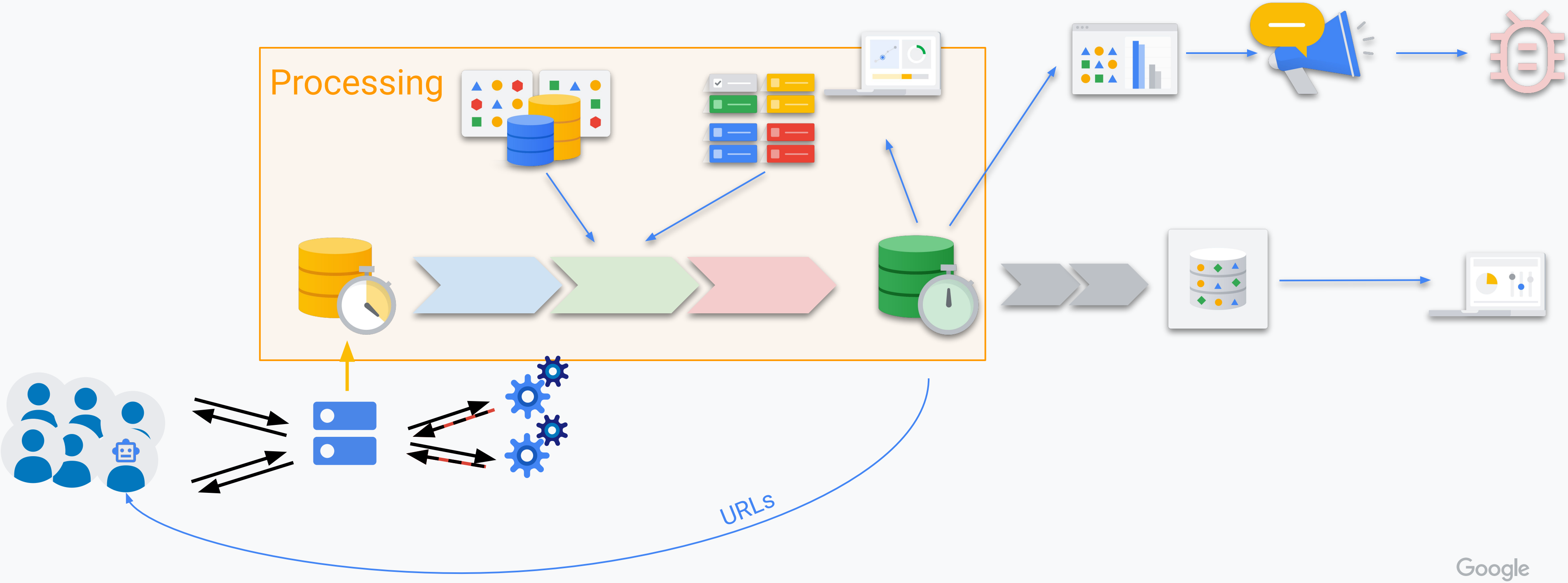
This context is crucial for streamlining **remediation efforts** and **automated bug filing.**

Risk signals provide data necessary to assess risk and prioritize according to it, e.g. sensitivity of the hosting domains based on Domain Tiers, exposure of services, volume of traffic.

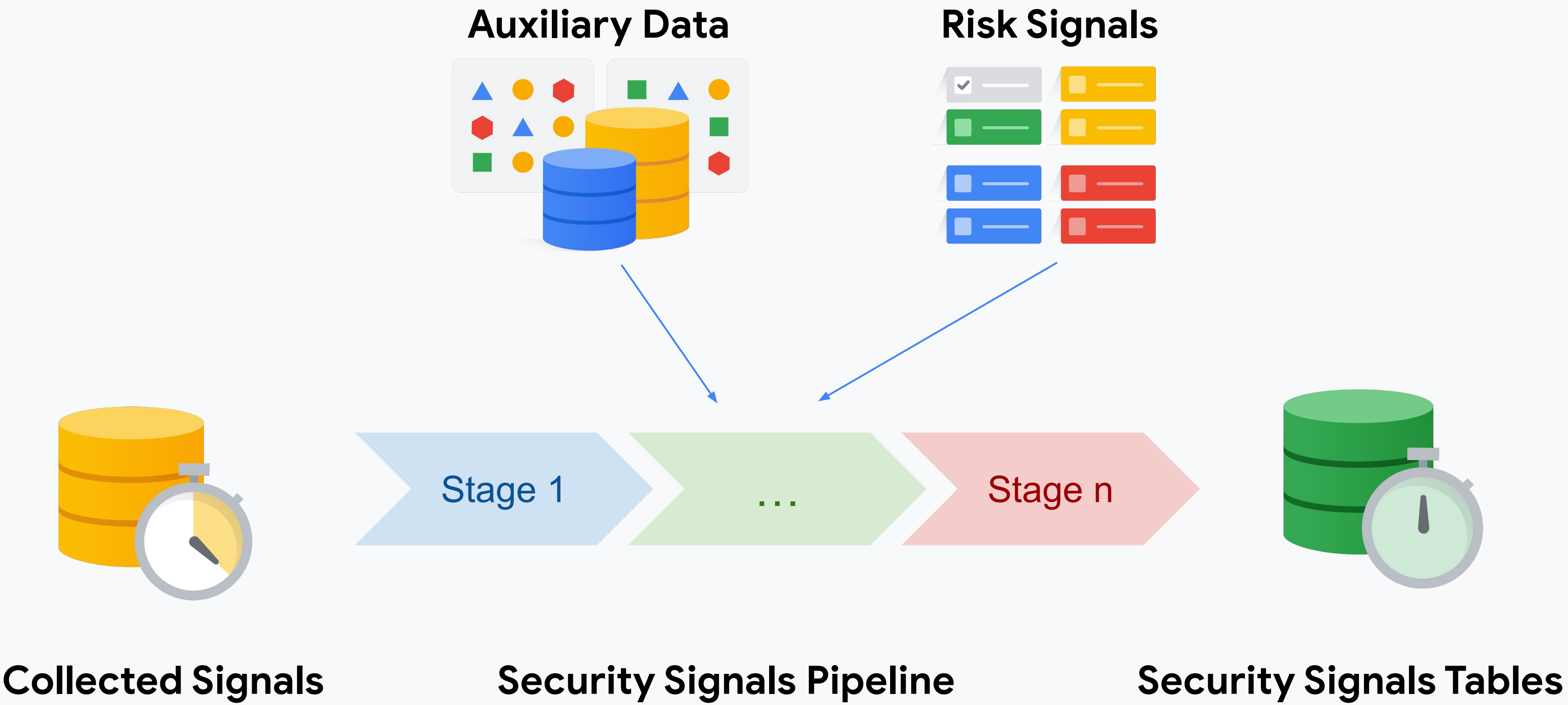
03

Processing Signals

Security Signals Architecture



Security Signals Pipeline



Auxiliary Data

Risk Signals

Stage 1

. . .

Stage n

Collected Signals

Security Signals Pipeline

Security Signals Tables

Security Signals Pipeline

Security Signals Pipeline is a Flume distributed map-reduce data processing pipeline, which:

- reads billions of collected signals,
- reduces their number by deduplication and initial evaluation,
- joins them with auxiliary data and risk signals,
- evaluates enriched signals to generate insights,
- persists them in Security Signals Tables.

The pipeline is heavily focused on **reducing the cardinality** of input data and **removing privacy sensitive information**, and producing **high-quality output**, which can be queried efficiently.

Cardinality Reduction

Collected Security Signals have billions of entries with high-cardinality dimensions, which makes them impractical to query. The pipeline reduces cardinality by aggregating values, while maintaining data usefulness.

URL paths often contain superfluous information, e.g. capability-bearing tokens, timestamps, user inputs. All URL paths are **redacted** into *path patterns* by:

1. Leveraging path routing information to match and replace variable parts, e.g. from synthetic signals or per-service infrastructure configurations (API definition).

2. On remaining paths, using filtering rules based on a manually curated set of well-known high-entropy paths.

3. On the left-over paths, executing a ML model (random forest of 11 trees with max depth of 5).

Security Signals Tables

Output of the Pipeline is the main source of data and insights needed by Security Signals.

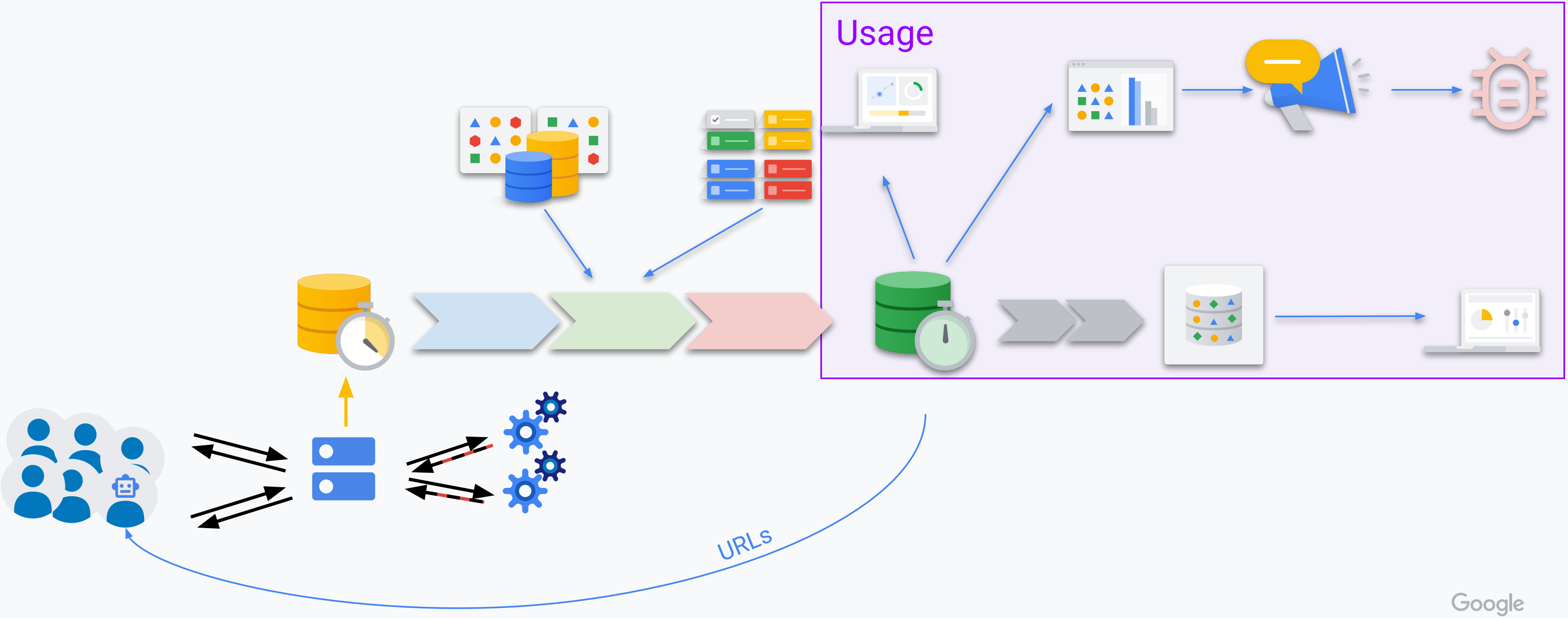
It is:

- Persisting only aggregated and de-identified data,
- Accessed by approved engineers and job roles,
- Created from previously collected data every day,
- Monitored to detect any anomalies in quality of data,
- Retained for 30 days.

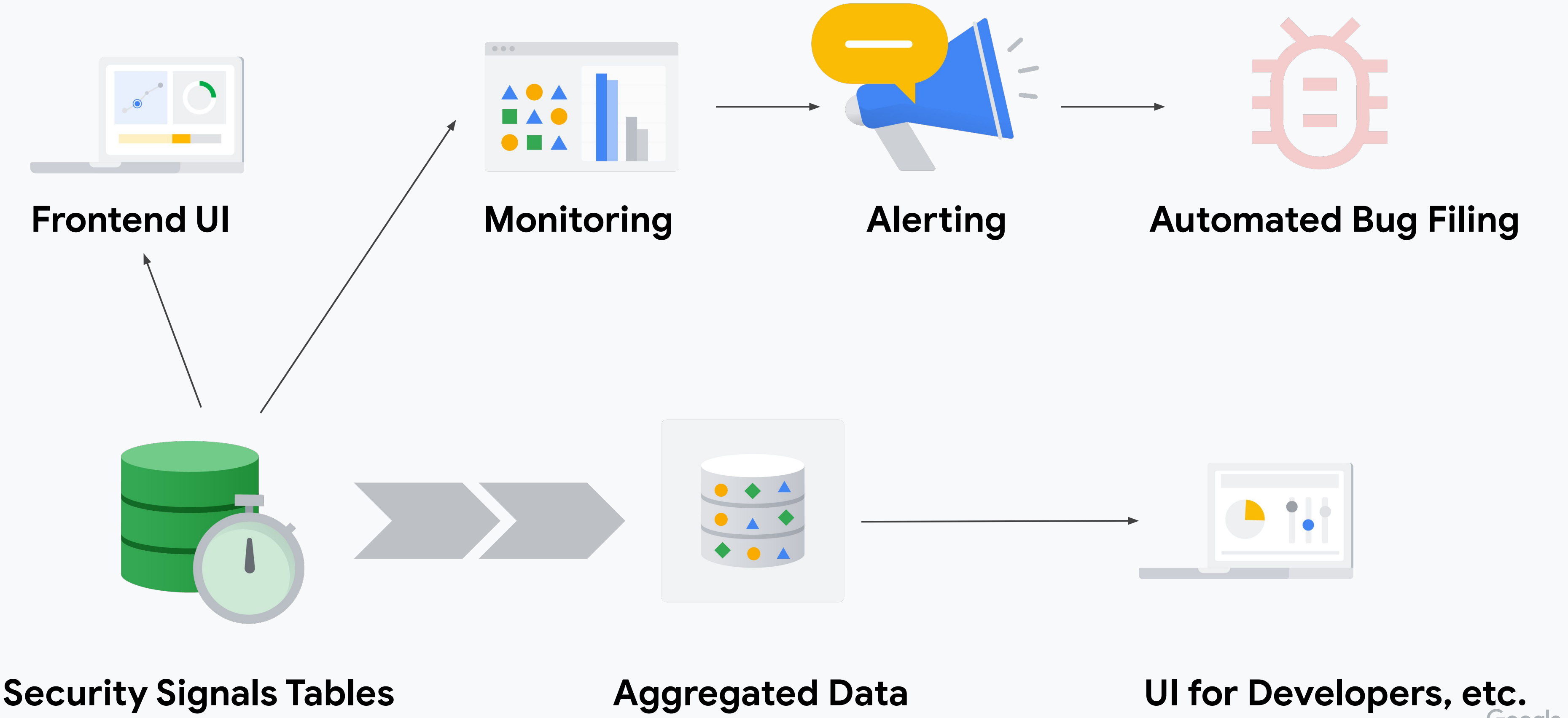
04

Using Data to Improve Security

Security Signals Architecture



Security Signals Tables Users



Frontend UI

Monitoring

Alerting

Automated Bug Filing

Security Signals Tables

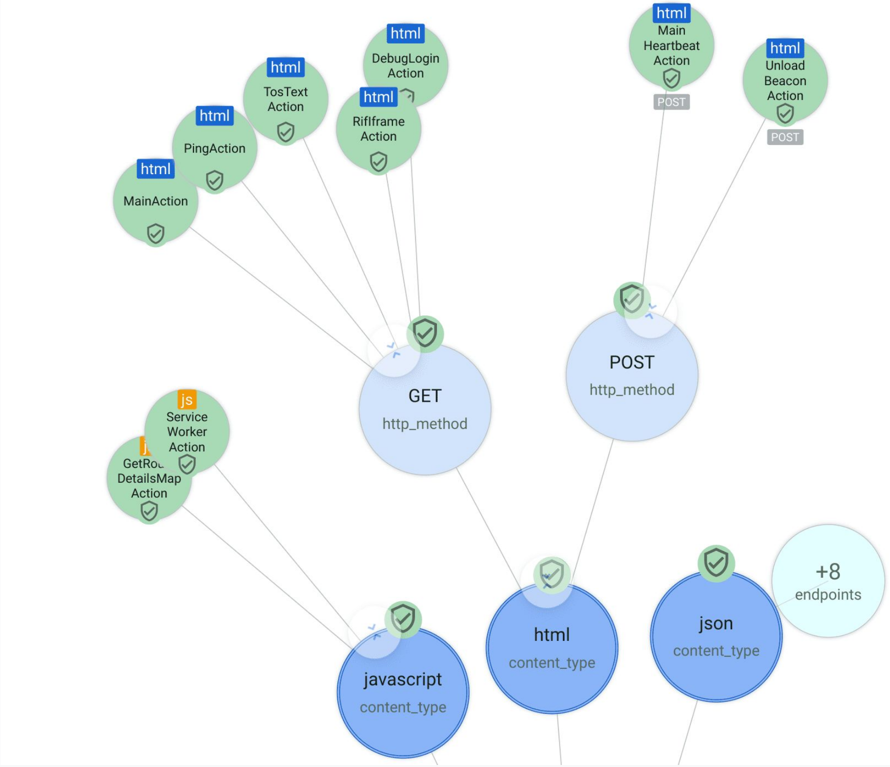
Aggregated Data

UI for Developers, etc.

Security Signals UI for Security Engineers

Application endpoints are presented as interactive “bubbles” organized by code package and color-coded to reflect their security status. This helps:

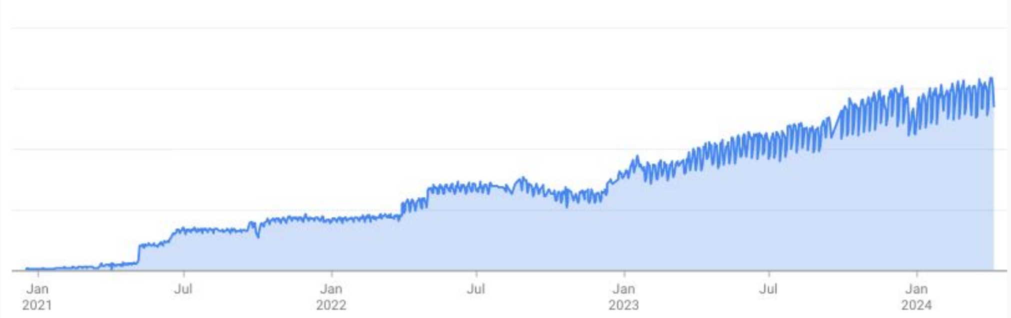
- Identifying security gaps,
- Initiating targeted remediations,
- Filing pre-populated bugs.



Monitoring, Alerting, Bug Filing

Continuous monitoring of Security Signals Tables allows:

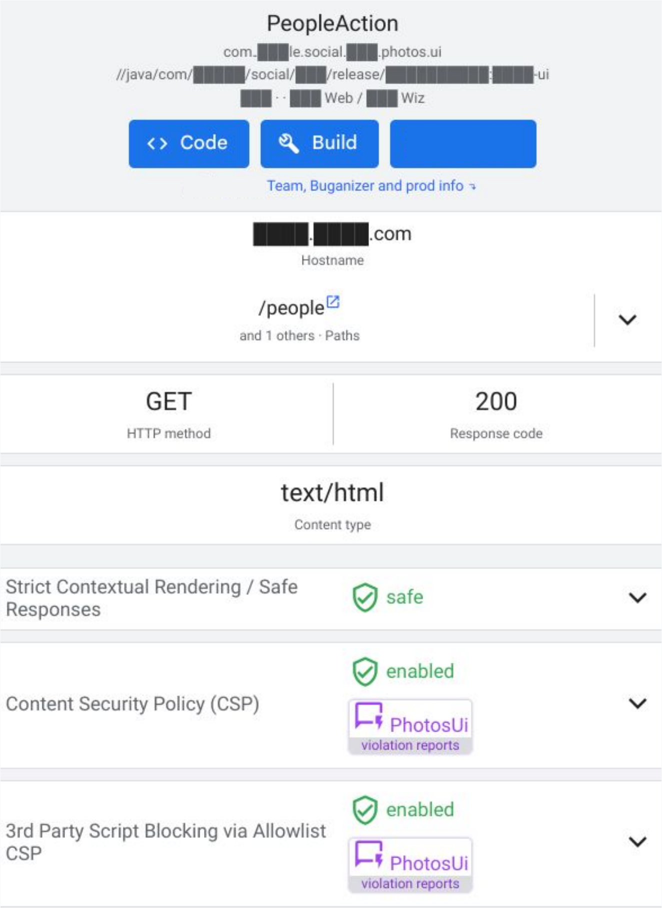
- Monitoring progress regarding coverage of security mitigation measures,
- Identifying violations of predefined security invariants,
- Monitoring regressions,
- Alerting about anomalies, findings and regressions,
- Automatically filing and assigning bugs for high confidence findings by leveraging ownership information within Security Signals.



Web Security Portal for Product Engineers

Web Security Portal provides insights tailored to each team’s application framework. The portal:

- is dedicated to developers without security expertise,
- shows web security posture of a product,
- highlights areas for improvement,
- offers framework-specific recommendations.



PeopleAction

com.[redacted]le.social.[redacted].photos.ui

//java/com/[redacted]/social/[redacted]/release/[redacted]:[redacted]-ui

[redacted] · · [redacted] Web / [redacted] Wiz

[redacted].[redacted].com

Hostname

/people

and 1 others · Paths

GET HTTP method	200 Response code

text/html

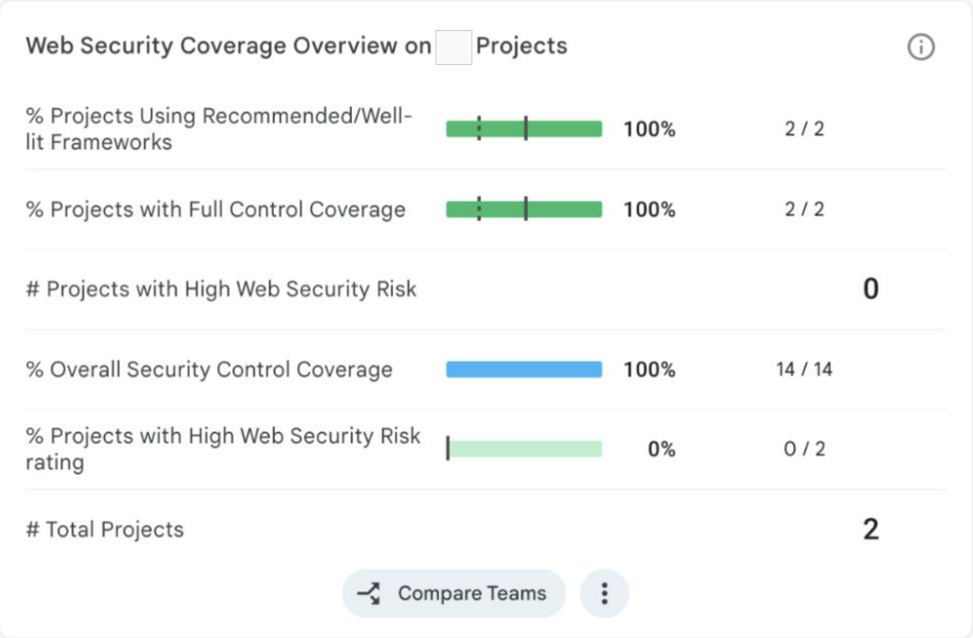
Content type

Strict Contextual Rendering / Safe Responses	✓ safe
Content Security Policy (CSP)	✓ enabled PhotosUi violation reports
3rd Party Script Blocking via Allowlist CSP	✓ enabled PhotosUi violation reports

Dashboards for Executives

Security Signals provides high-level visibility and strategic insights to executives to allow:

- Assessing overall web security posture,
- Identifying areas of focus,
- Tracking progress and quantifying impact,
- Risk-based prioritization,
- Optimizing resource allocation decisions.



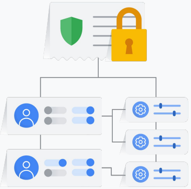
Web Security Coverage Overview on □ Projects

% Projects Using Recommended/Well-lit Frameworks	100%	2 / 2	
% Projects with Full Control Coverage	100%	2 / 2	
# Projects with High Web Security Risk			0
% Overall Security Control Coverage	100%	14 / 14	
% Projects with High Web Security Risk rating	0%	0 / 2	
# Total Projects			2

05

Use Cases

Safe Coding: Security Engineering Use Cases



The responsibility for ensuring security is moved to the developer environment (**Safe Coding environment**) and product design (**secure-by-design**) and includes:

- Hardened and secure-by-design web frameworks,
- Frontend guidelines and recommendations,
- Required web security features.

New web applications adopt this approach seamlessly, but architecture of existing ones need to be adjusted.

Use Case: Security Research & Remediations

Legacy code and systems create the **need to continuously improve the security** state of existing web services.

Security **remediations** are engineering efforts aimed at mitigating systemic sources of vulnerabilities. Each crucial step of remediations is driven by Security Signals:

1. Identifying potential security risks.
2. Designing mitigations.
3. Adopting mitigations and detecting future regressions (next slides).

Use Case: Adoption of Web Security Mitigations

Identified and evaluated classes of vulnerabilities are then addressed by proposed mitigations at scale, by:

- **Identifying** services or specific endpoints that benefit from the mitigation,
- Gradually **rolling out** new security mitigations,
- **Tracking** deployment progress across hundreds or thousands of services,
- **Handling exceptions** and special cases,
- **Alerting** on any regression.

... and all that without impacting the functionality of any service.

Example (groups of) **mitigations**: Content Security Policy, Trusted Types, Fetch Metadata isolation policies, Cross-Origin Opener Policy, etc.

Use Case: Additional Capabilities

< ai>...</ai>

- **JavaScript Signals pipeline** for all executed JavaScript scripts.
- **Improving Security Scanning Coverage**, which is limited by crawling.
- **Non-security Use Cases** to monitor rollouts of web features, debug issues, etc. (~50 teams across Google).
- **Surfacing AI/ML Properties** by Web Endpoints.

06

Example

Example: Cross-Site Request Forgery

Webpages can include resources from other places, e.g.

```
<img src="https://example.com/images/cat.jpg" alt="some cats"/>
```

... or turn off your home router:

```
<img src="http://192.168.0.1/off.php"/>
```

... or transfer money:

```
<form action="https://mybank.com/send?amount=10k&from=thomas&to=eve&do=true"
       method="POST" id="form">
</form>
<script>document.getElementById('form').submit()</script>
```

Example: Cross-Site Request Forgery (Prevention)

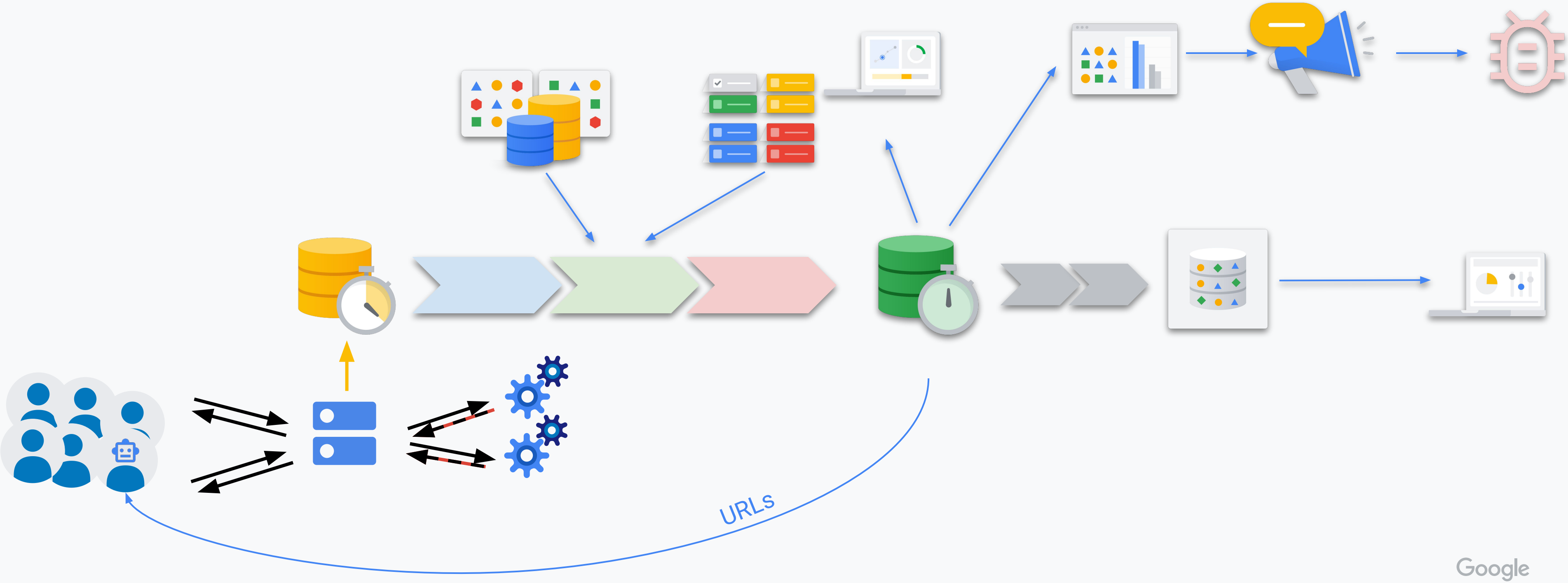
CSRF/XSRF token: a new piece of information that is both **unguessable** and **client-correlated** and send with each request.

```
Xsrf-token=YL9yaTsbfn
```

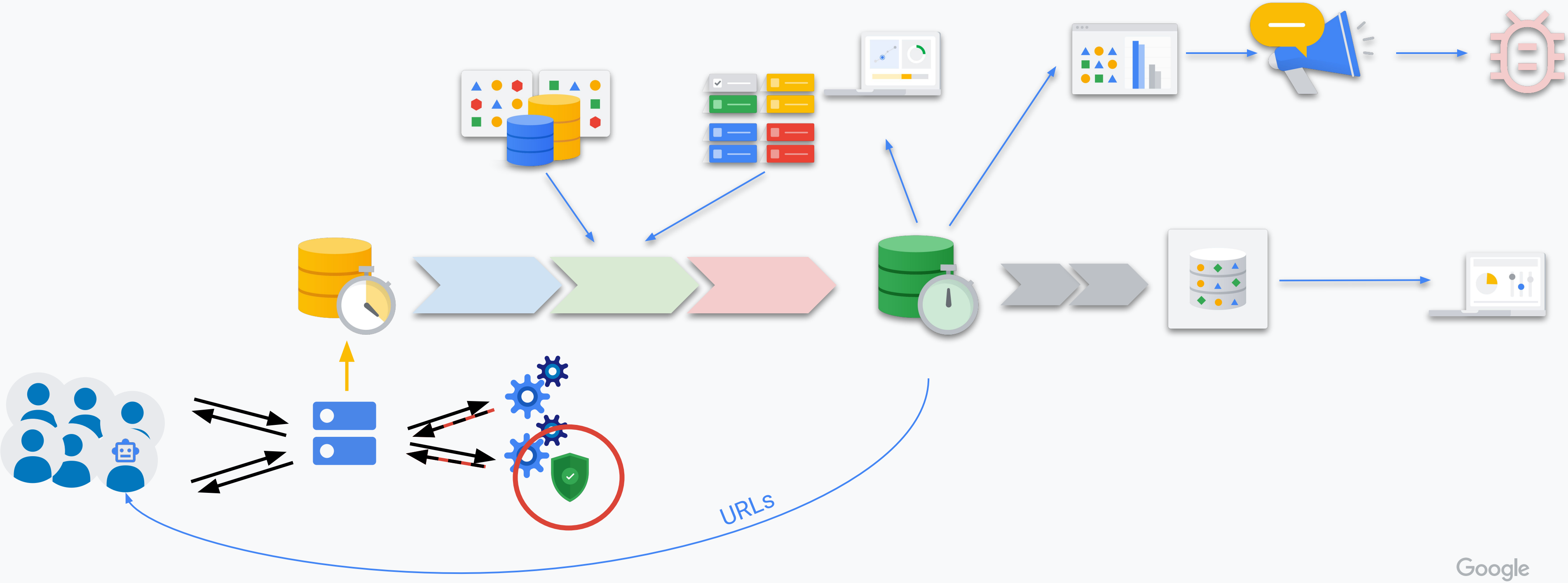
The rollout:

1. Identify URL endpoints implementing state-changing functionality and their XSRF tokens.
2. Introduce a new synthetic security signal: `CSRF`.
3. Refactor web frameworks to populate `CSRF` signal, prioritizing them by Domain Tiers.
4. Handle exceptions/special cases.
5. Go to (3).

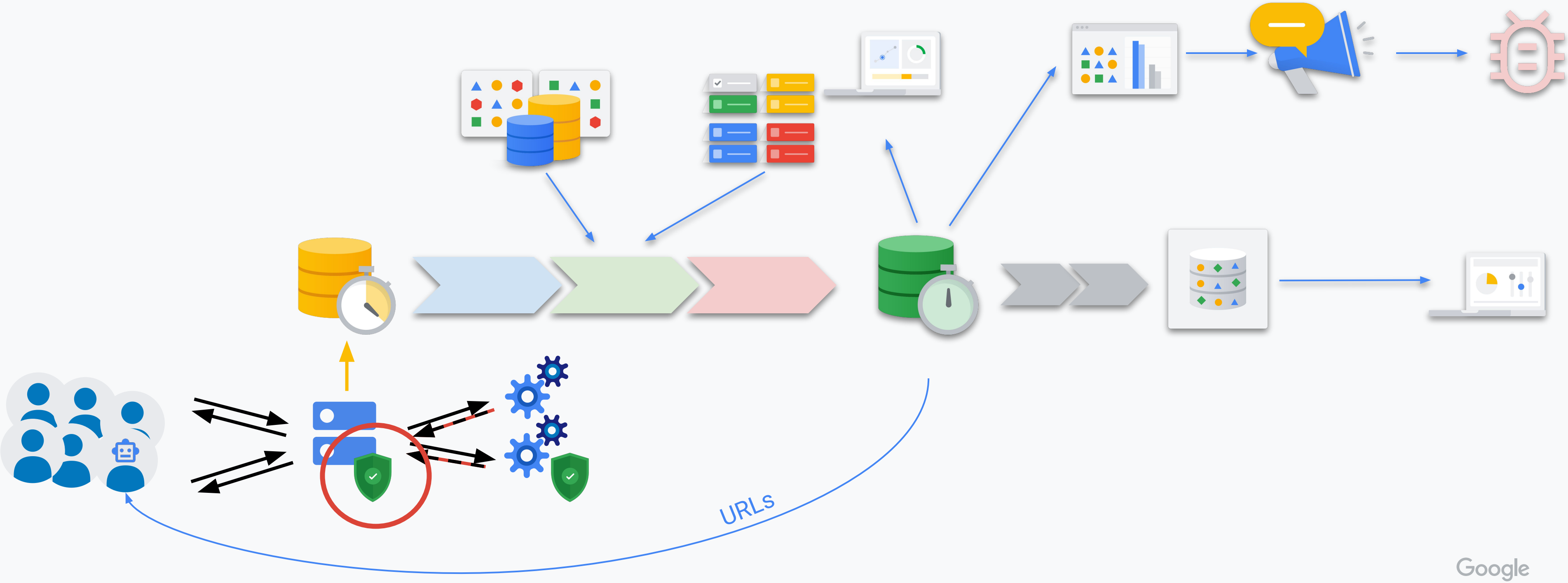
Example: Cross-Site Request Forgery (Data Flow)



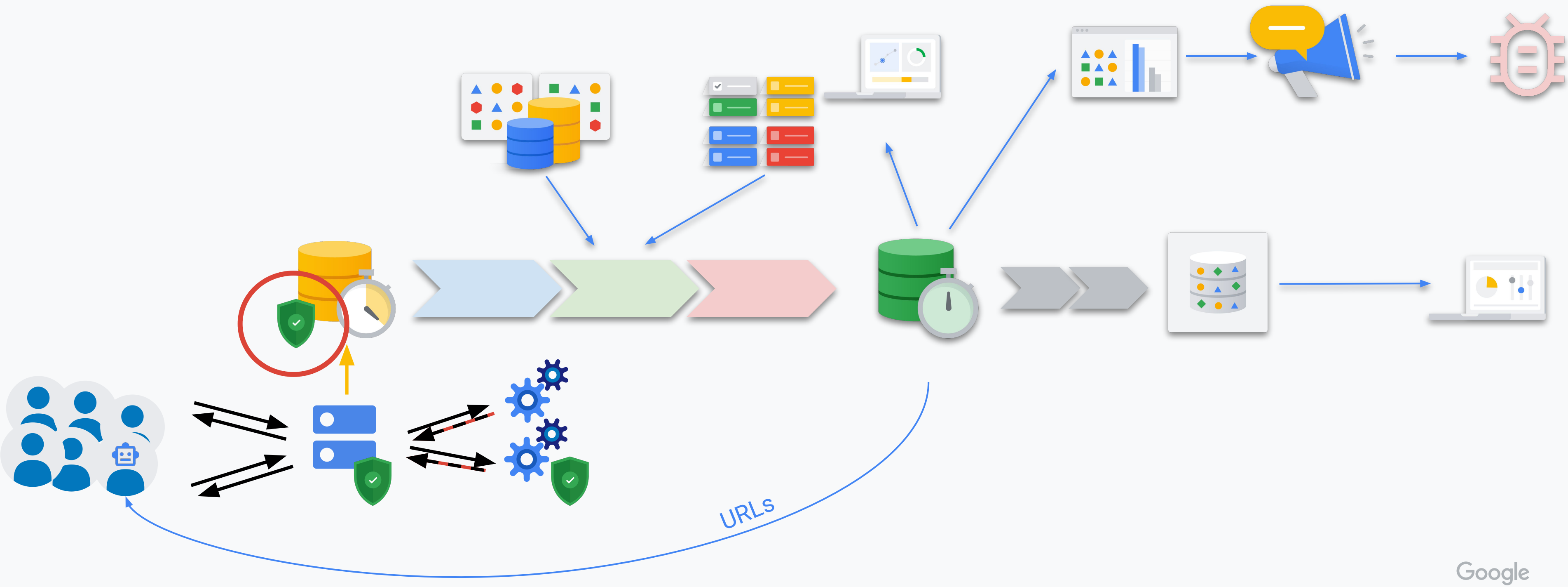
Example: Cross-Site Request Forgery (Data Flow)



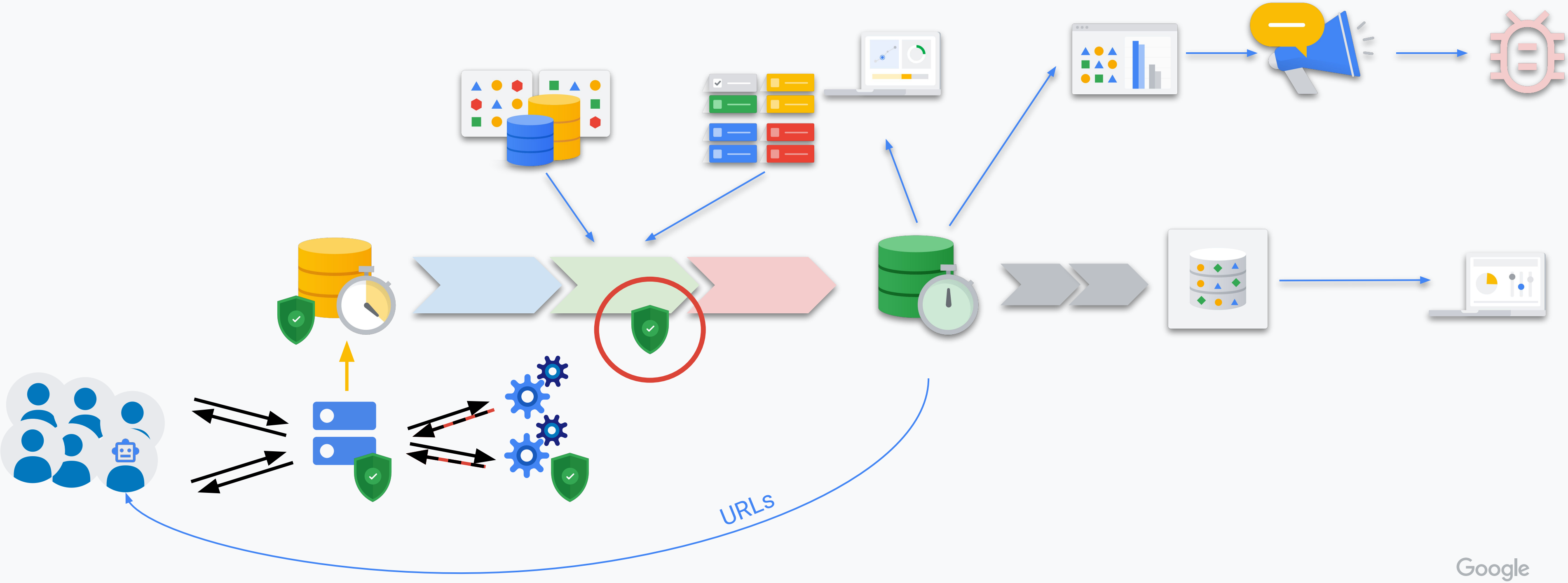
Example: Cross-Site Request Forgery (Data Flow)



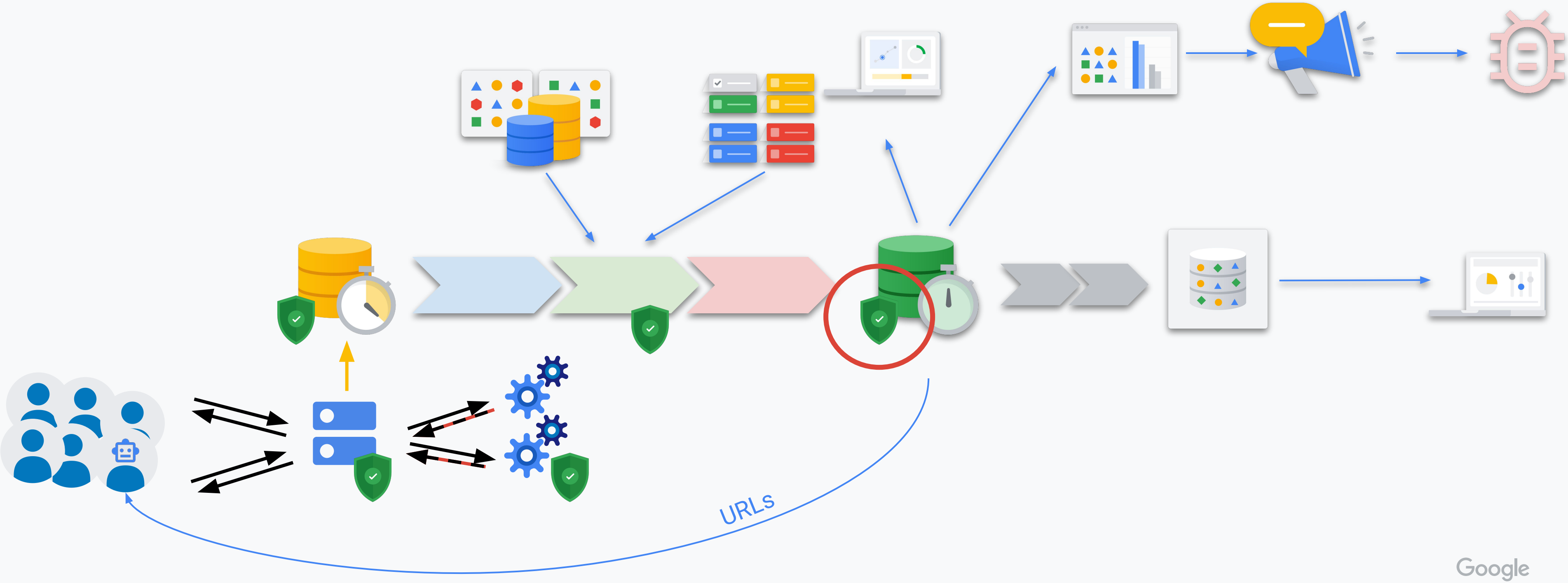
Example: Cross-Site Request Forgery (Data Flow)



Example: Cross-Site Request Forgery (Data Flow)

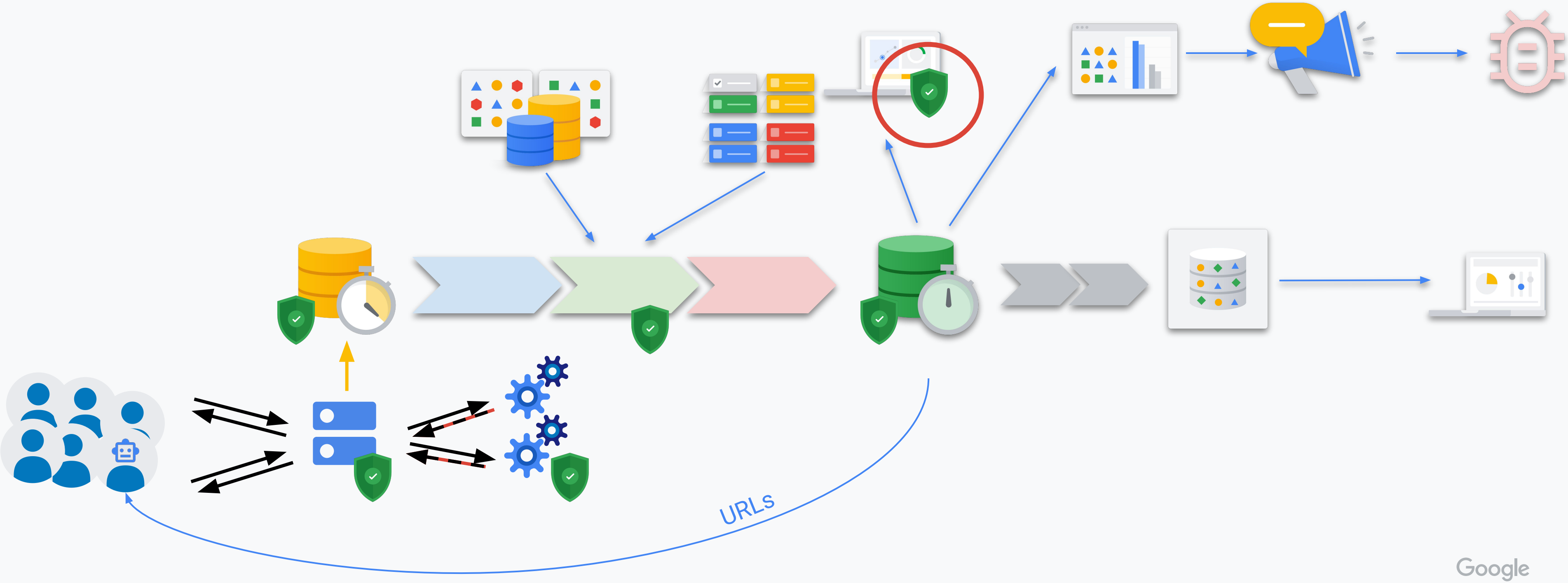


Example: Cross-Site Request Forgery (Data Flow)

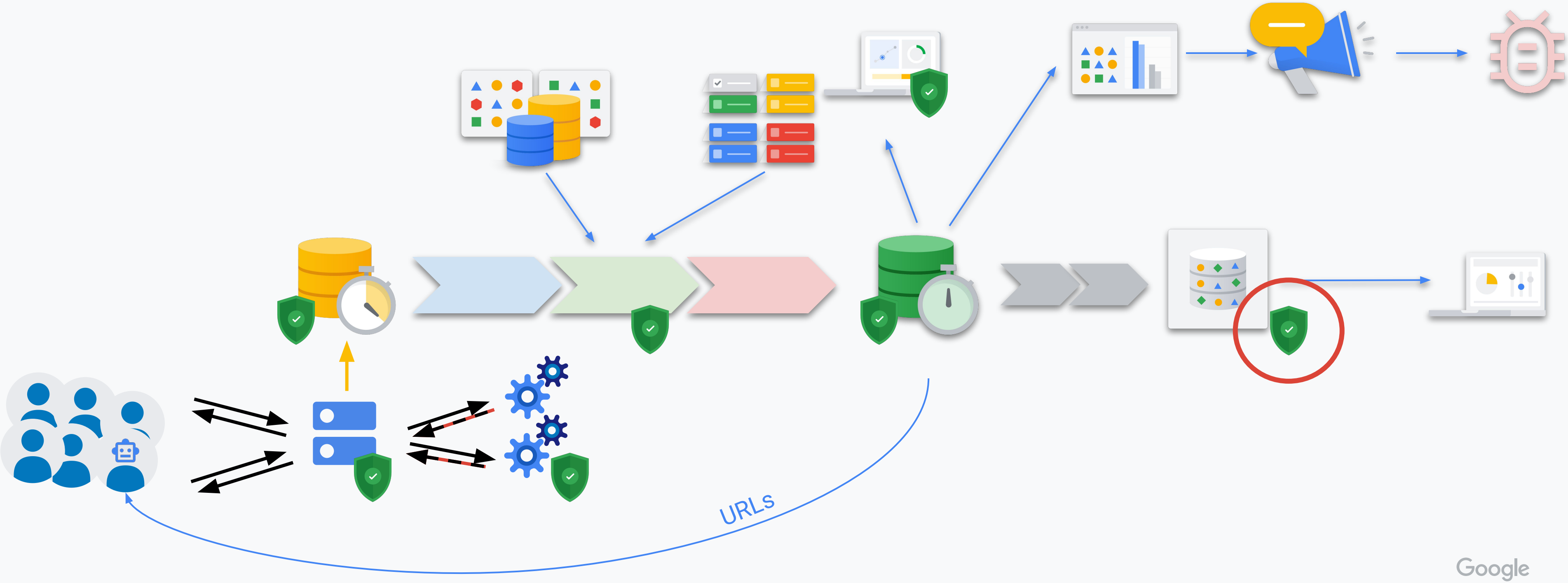


Google

Example: Cross-Site Request Forgery (Data Flow)

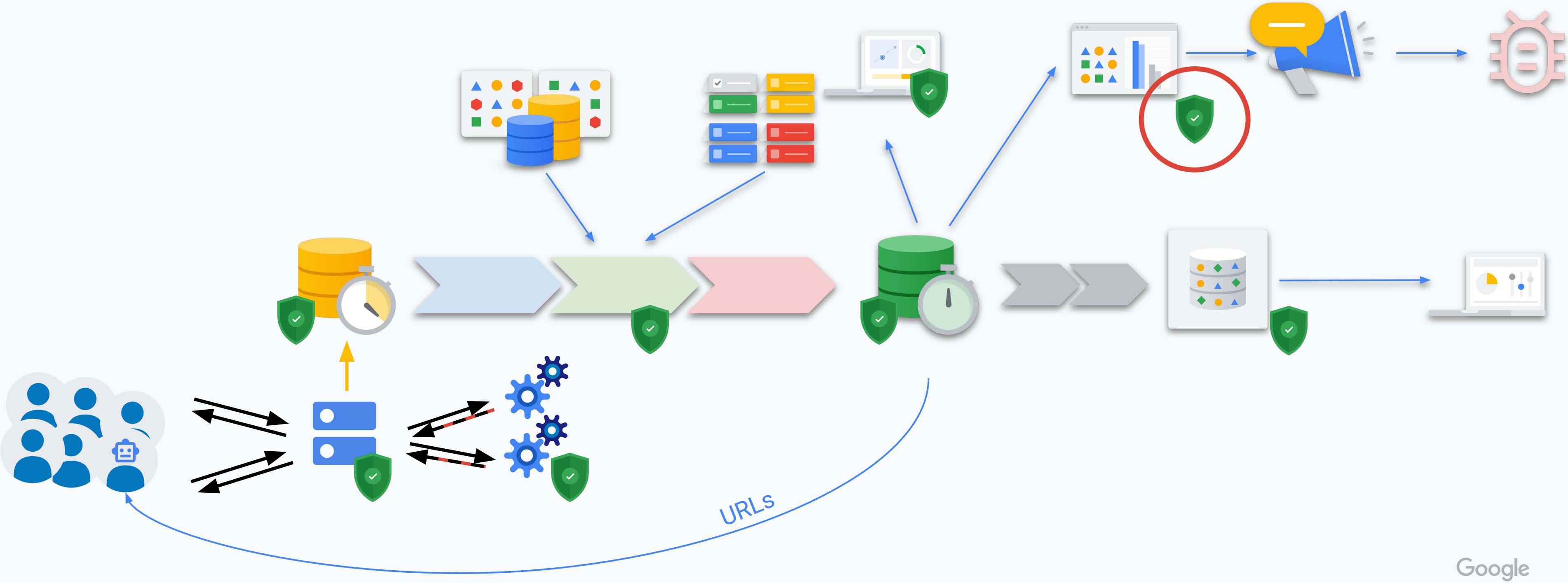


Example: Cross-Site Request Forgery (Data Flow)



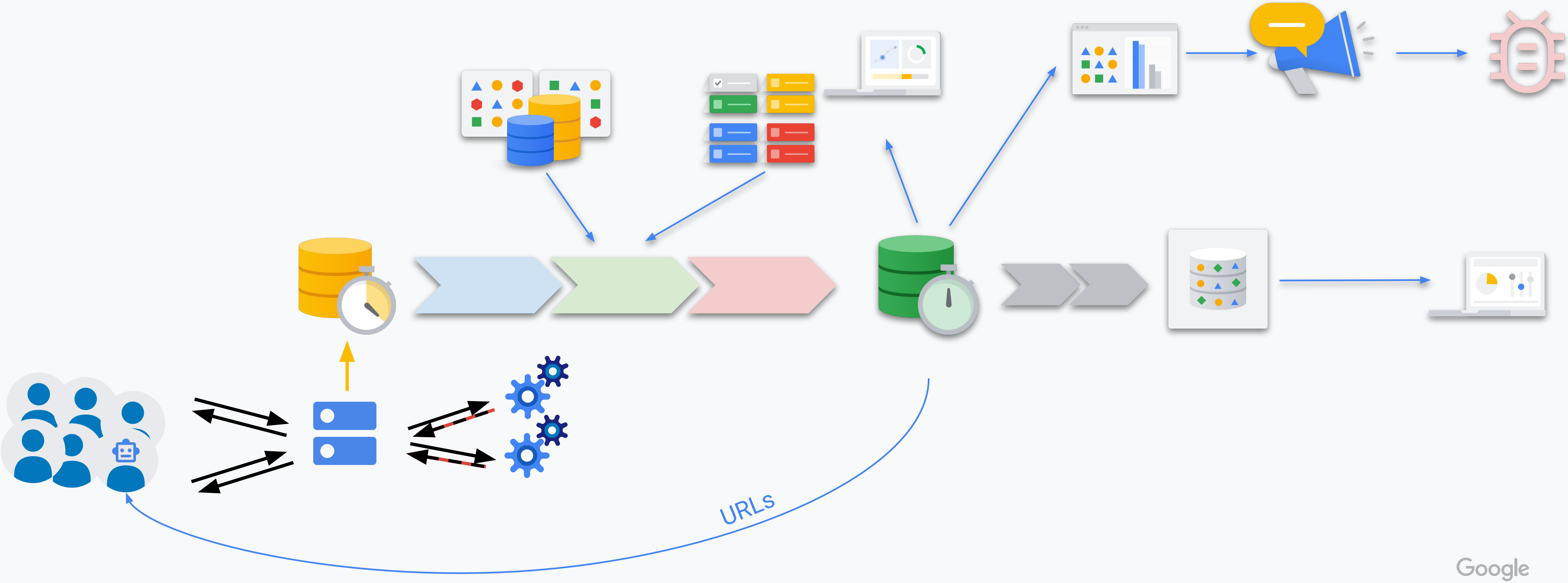
Google

Example: Cross-Site Request Forgery (Data Flow)



Google

Security Signals Infrastructure



Thank you