Evaluating Cryptographic API Misuse Detectors for Go

Vivi Andersson, Martin Monperrus
KTH Royal Institute of Technology, Stockholm

April 18, 2026
Rio de Janeiro, Brazil
SVM 2026

Cryptographic API Misuse

Definition.

Incorrect usage of a cryptographic API that breaks the security guarantees the API is meant to provide.

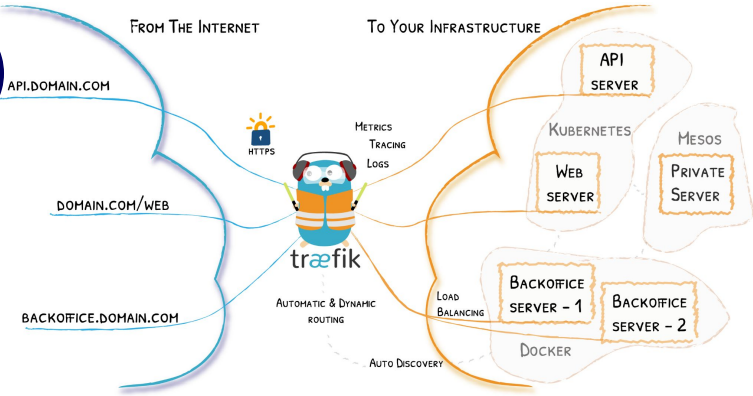
img: freepik

Example: Traefik (CVE-2025-66491)

TLS certificate verification config inverted

```
annotations:
  nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
  nginx.ingress.kubernetes.io/proxy-ssl-verify: "on"
  nginx.ingress.kubernetes.io/proxy-ssl-secret: "<namespace>/<secret>"
```

User sets `proxy-ssl-verify` "**on**"



src: geek-cookbook

```
nst := &namedServersTransport{
    Name: provider.Normalize(namespace + "-" + name),
    ServersTransport: &dynamic.ServersTransport{
        ServerName:         ptr.Deref(cfg.ProxySSLName,
            ptr.Deref(cfg.ProxySSLServerName, "")),
        InsecureSkipVerify: strings.ToLower(ptr.Deref(cfg.ProxySSLVerify, "off")) == "on",
    },
}
```

`InsecureSkipVerify` evaluates to **true**

→ Backend TLS authentication disabled

→ Enables impersonation/MITM

Go + Crypto = ❤️ ?

- Mature and strong native crypto suite! (`crypto/ssh`, `x/crypto/`..)

- Crypto misuses reports
  - CVE-2024-45337 in `golang.org/x/ssh`
    - misuse of PublicKeyCallback → authorization bypass
  - CVE-2024-51744 in `golang-jwt/jwt` (JSON Web Tokens)
    - ParseWithClaims error handling → accept invalid tokens
  - …

- Go is a major stack for security-critical infrastructure: Let's Encrypt's Boulder (CA), Kubernetes (orchestration), Traefik (reverse proxy), Terraform (IaC)…

Prior Work

- **Java.** Comparative studies of crypto misuse detectors [1-3] show disagreement between tools and frequent developer rejection of alerts
- **Python.** Evaluations in Python [4-5] emphasize that cryptographic misuse patterns differ across language ecosystems
- **Go.** Go-specific detectors have been proposed: **CryptoGo** [6] introduced the first approach, **Gopher** [7] extendeds it, yet no comparison with existing tools

[1] Zhang, Ying, et al. “Automatic Detection of Java Cryptographic API Misuses: Are We There Yet?” IEEE TSE 2022
[2] Chen, Yikang, et al. “Towards Precise Reporting of Cryptographic Misuses.” NDSS, Internet Society, 2024
[3] Ami, Amit Seal, et al. “Why Crypto-detectors Fail: A Systematic Evaluation of Cryptographic Misuse Detection Techniques.” IEEE SP, 2022
[4] Wickert, Anna-Katharina, et al. “Python Crypto Misuses in the Wild.” ESEM, 2021
[5] Frantz, Miles, et al. “Methods and Benchmark for Detecting Cryptographic API Misuses in Python.” IEEE TSE, 2024
[6] Li, Wenqing, et al. “Cryptogo: Automatic Detection of Go Cryptographic API Misuses.” IEEE ASAC, 2022
|7] Zhang, Yuexi, et al. “Gopher: High-Precision and Deep-Dive Detection of Cryptographic API Misuse in the Go Ecosystem.” ACM CCS 2024

Problem

Cryptographic API misuses keep reaching production in Go systems that powers critical infrastructure like Kubernetes and Traefik.

Detectors exist, but no one has compared what they catch or what they miss.

This work compares four Go crypto misuse detectors across 328 projects and 14 misuse classes

RQ1. What is the design space of crypto misuse tools in Go?

We survey capabilities of the tools and consolidate a taxonomy of 14 Go crypto misuse classes

Sourced Detectors

CodeQL

GitHub • Open source

Datalog queries over a fact database, with data-flow reasoning.

Gosec

Community • Open source

SSA for some rules, AST pattern matching for others.

Gopher

Academic • Binary-only

gopher

SSA-based taint tracking, with dynamically updatable rules.

Snyk Code

Commercial • Proprietary

snyk

Static analysis product; implementation details undisclosed.

RQ1: Taxonomy and Tool Coverage

Category	#	Rule	CodeQL	Gopher	Gosec	Snyk
Primitives	01	Insecure algorithms		✓	✓	✓
	02	Insecure PRNG	✓	✓	✓	✓
	03	Deprecated Go function		✓		
Key mgmt	04	Constant/predictable key		✓		
	05	Short key length	✓	✓	✓	✓
	06	Static/predictable IV		✓	✓	
PBKDF	07	Short salt length		✓		
	08	Predictable salt		✓		
	09	Low hash iterations		✓		
Transport	10	HTTP protocol		✓		
	11	TLS/SSL issues	✓	✓	✓	✓
SSH	12	Insecure SSH suite		✓		
	13	No host key validation	✓	✓	✓	
Token	14	No JWT verification	✓			

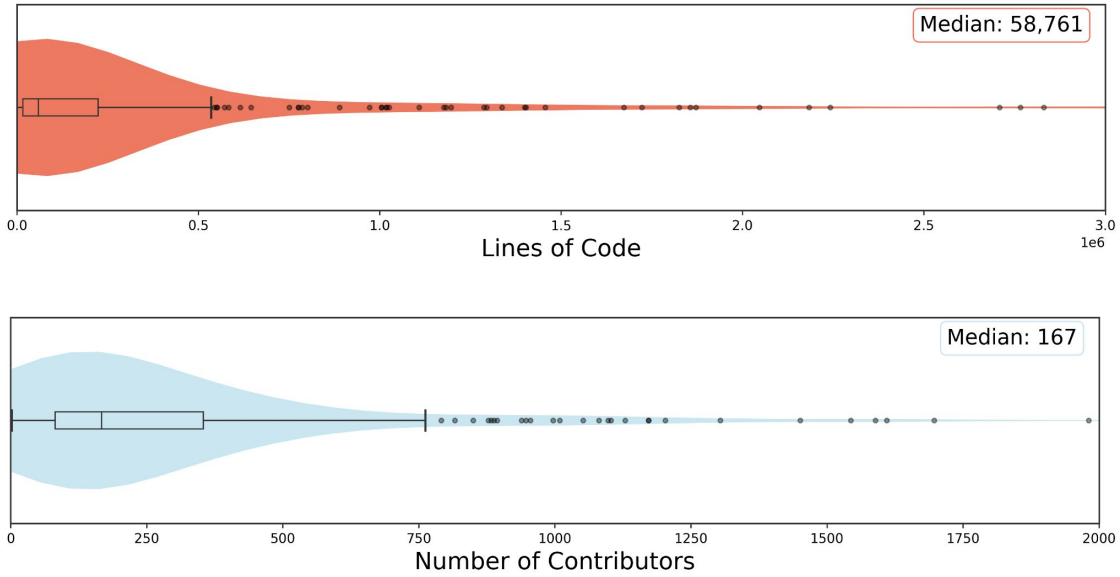
Takeaway. Tool support is uneven: only 3 of 14 misuse classes are covered by all four tools

RQ2. To what extent are crypto misuse detection tools for Go consistent with each other?

We run all four tools on 328 security-relevant Go projects and compare their findings

Dataset

- **328** open-source Go repositories, filtered from a set of 920 popular projects [8]
  - Actively **maintained** (commit within the last 6 months, has stable release)
  - Protocols **relying on cryptographic security**, e.g., infrastructure, networking, microservices



[8] Chen et al. *An Empirical Study of Cgo Usage in Go Projects: Distribution, Purposes, Patterns and Critical Issues.* JSS (2025) 112601

RQ2: Tool Execution Success

> Do they successfully analyse the projects?

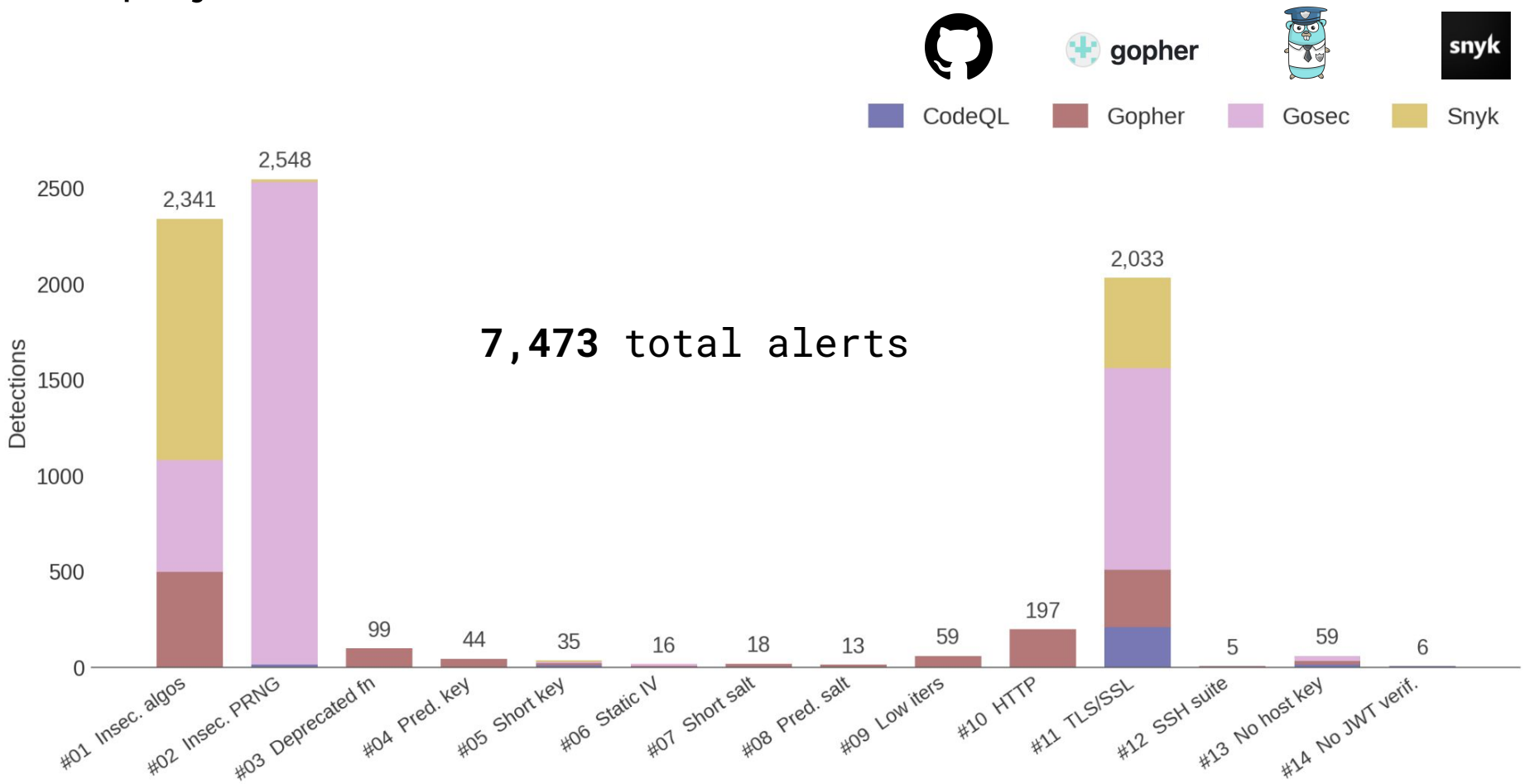
	CodeQL	Gopher	Gosec	Snyk
Projects analyzed (%)	91.8	77.1	100	100
With findings (% of analyzed)	30.9	64.8	84.5	92.7
With findings (% of dataset)	28.4	50.0	84.5	92.7

Takeaway. Practical usability differs: Gosec and Snyk run reliably, while Gopher and CodeQL fail on a non-trivial fraction of projects.

RQ2: Alert Magnitude

Takeaway. Excessive detection counts for #01, #02, #11 indicates aggressive, insensitive alerts

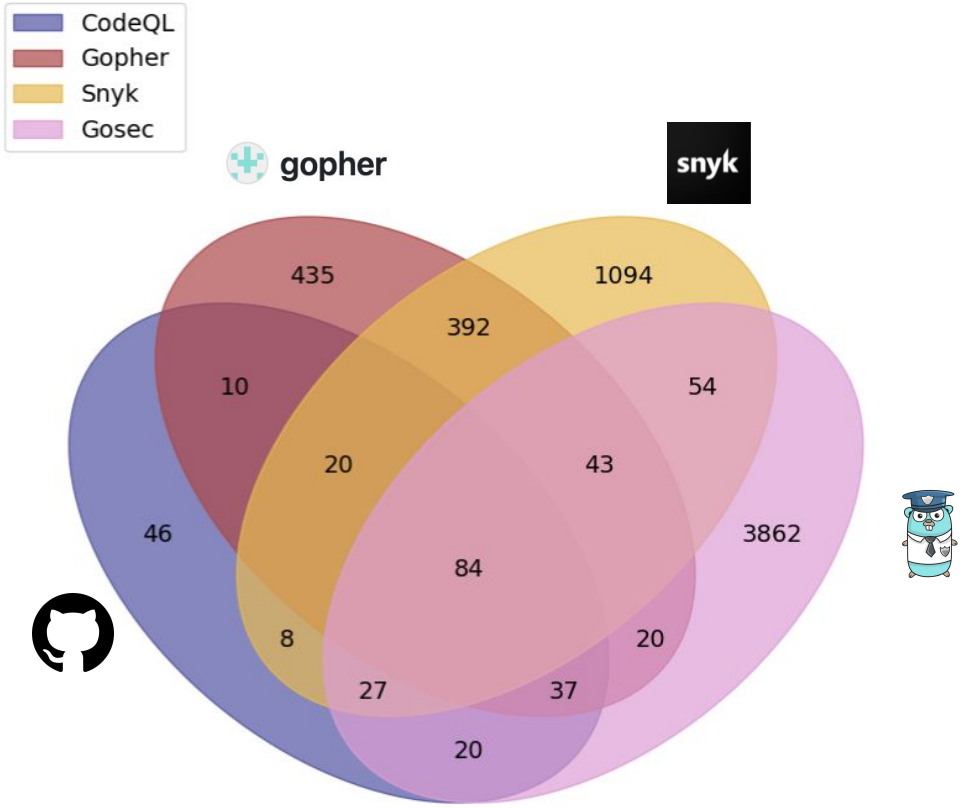
> Across 328 projects



RQ2: Line Level Agreement

- **1.3%** (84) are flagged by **all four** tools **11.6%** (715) are flagged by **at least two**

6,152 uniquely flagged lines

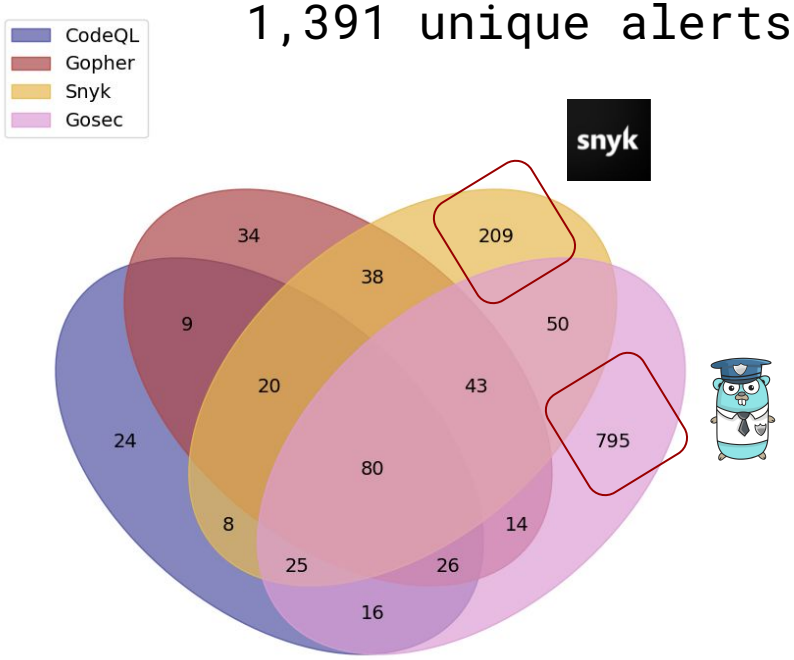


Takeaway. The 4-way agreement alerts are a practical starting point for triage.

#11 SSL/TLS Issues

> Outdated **TLS versions** (<1.2), insecure **cipher suites** (CBC-based), or skipped **certificate verification** (`InsecureSkipVerify`)

`1,391 unique alerts`



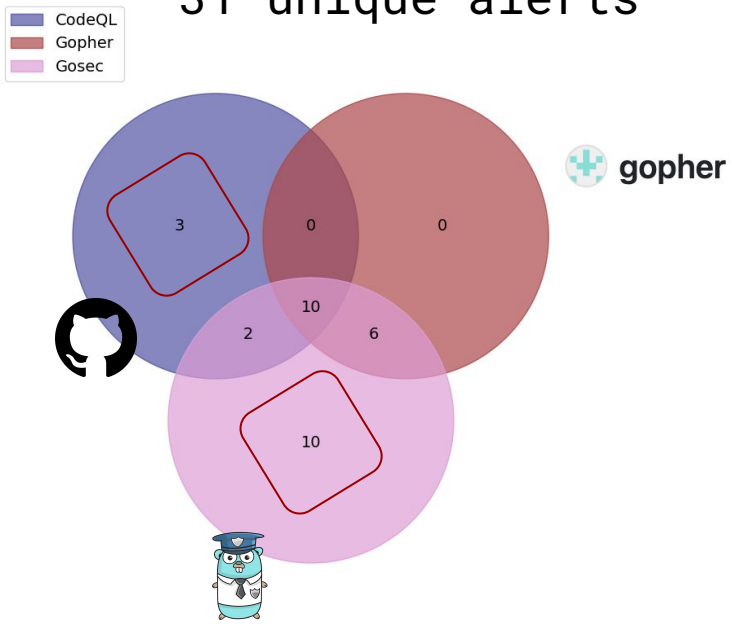
- Manual samples
  - **Gosec-only**: 3 of 5 are false positives (flags tls.Config without checking configured values)
  - **Snyk Code-only**: no outright FPs, but low severity

Takeaway. Gosec is noisier, while Snyk Code's sampled alerts were generally valid, but not necessarily security issues

#13 No Host Key Validation (SSH)

> *SSH client accepts any server without verifying its identity (`InsecureIgnoreHostkey`)*

31 unique alerts



- **Gosec.** Pattern-matches `InsecureIgnoreHostKey`

```
HostKeyCallback: ssh.InsecureIgnoreHostKey(),  // TODO validate server before?
    // scaleway.go:327
```

- **CodeQL.** Detects semantic equivalents (return nil, accept every key).

```
HostKeyCallback: func(hostname string, remote net.Addr,
    key ssh.PublicKey) error {
    return nil  // accept any host
},
```

Takeaway. CodeQL covers semantic equivalents that other tools miss.

Areas for Improvement

Sensitivity to Usage Context

Improvement. Future tools should assess an API's security role and when its use becomes risky.

#05 short key size

```
func hashFile(name string, maxSize uint64, hashType ...HashType) (
    nameToHash map[HashType]Digest, nbytes uint64, err error,
) {
    // ...
    h, _ := blake2b.New512(nil)
    // ...
}
```

Used for content hashing, not as a MAC

elastic/beats

#13 no ssh hostkey validation

```
var hostKeyCallback ssh.HostKeyCallback
if kn := os.Getenv("SSH_KNOWN_HOSTS"); kn != "" {
    hostKeyCallback, err = knownhosts.New(kn)
} else {  // no known_hosts configured
    hostKeyCallback = ssh.InsecureIgnoreHostKey()  // compatibility fallback
}
```

Insecure usage only exists in compatibility code, requiring explicit configuraiton

juicedata/juicefs

Improved Path Sensitivity

Improvement. Tools need improved **path sensitivity** and side-effect modeling to recognize guards around the flagged sink.

Flaga a sink, but misses missing preceding or subsequent guards

#05 short key size

unreachable

flagged

```
func (kr *KeyRequest) Generate() (crypto.PrivateKey, error) {
    switch kr.Algo() {
    case "rsa":
        if kr.Size() < 2048 {
            return nil, errors.New("RSA key is too weak")
        }
        if kr.Size() > 8192 {
            return nil, errors.New("RSA key size too large")
        }
        return rsa.GenerateKey(rand.Reader, kr.Size())  // ← sink
    }
}
```

cloudflare/cfssl

#11 TLS min version too low

```
func setTLSDefaults(ctls *tls.Config) {  // tls.go:14
    ctls.MinVersion = tls.VersionTLS12
    ctls.MaxVersion = tls.VersionTLS13
    ctls.CipherSuites = []uint16{ ... }
}

tlsConfig := &tls.Config{Certificates: []tls.Certificate{cert}, RootCAs: roots}
setTLSDefaults(tlsConfig)  // tls.go:98
```

flagged

safe defaults

coredns/coredns

Replication pkg

Contributions

The **first systematic analysis** of the crypto-misuse problem space for Go.

We analysed **4 detectors** on **328 security-critical** Go projects, identifying **7,473** misuses.

Low overlap and excessive alerts indicates an **ensemble approach** with focus on shared detections is a practical near-term strategy.