LEAK suite Shadow AI

Author: Çınar Yıldırım Date: Sep 13, 2026
Somebody pastes a customer record into ChatGPT. Who stops it?

This suite scores SolonGate's scanner against three real products on the same corpus, with the same rules, on the same machine. Nothing here is a baseline we wrote to lose.

tool	version	what it is
`solongate/sgdetect`	this tree	the product's own scanner, built in detectors, no project config
`presidio`	2.2.x	Microsoft Presidio, default 0.5 threshold, its own recognisers
`gitleaks`	v8	the secret scanner most teams already run in CI
`trufflehog`	3.97.4	the other one, verification off so a synthetic key still counts

```
./scripts/run-leak.sh
```

Results 2026-09-13

1,890 real pastes carrying 4,930 marked values, against 3,642 clean pastes (~1.1M tokens of real open source code).

tool	detected	in scope	plain	evasion	located	false alarms /10k tokens	ms/paste
solongate/sgdetect	**94.7%**	**94.7%**	**99.8%**	**93.0%**	**77.3%**	**0.25**	1.12
presidio	43.7%	77.0%	47.4%	42.7%	42.9%	1.34	19.7
trufflehog	6.7%	44.5%	8.0%	6.3%	2.5%	**0.02**	0.69
gitleaks	5.7%	38.2%	7.1%	5.4%	5.5%	0.10	0.09

Two things are being asked, and they are not the same.

Detected did the tool know the value was in there? That is the question the product answers: the extension blocks or warns on the paste, and for that it only has to know. *Located* can it point at the value? That is what **redaction** needs, and it is strictly harder: a value that exists only base64 encoded inside the paste has

no character range in the text the person typed, so a scanner can know it is there and still be unable to mask it.

The 94% / 77% gap is exactly that. SolonGate blocks 94% of leaks and can safely redact 77% of them. Every other number below is *detected* unless it says otherwise.

In scope counts only the kinds a tool claims to cover scoring gitleaks zero on a Turkish tax number it never claimed would be a straw man.

What the table says, plainly:

- SolonGate detects **more than twice** what the best general purpose alternative detects, and fourteen times what a secret scanner detects.
- It also raises **fewer false alarms than Presidio** 0.99 per 10k tokens against 1.34. Both halves at once, which is the only combination that means anything.
- 53 of SolonGate's 109 false alarms are **real author email addresses in open source files**, and Presidio reports the same 53. They are arguably correct detections of real emails that merely are not interesting.
- Secret scanners are not DLP, and their precision is real: trufflehog raises 2 false alarms in 1.1M tokens. It also sees 7% of what actually leaves a machine, because a pasted customer spreadsheet is not a credential.

Where the advantage comes from

kind	solongate	presidio	gitleaks	trufflehog
tckn (national id)	**795/795**	135/795 *		
vkn (tax number)	**330/330**	29/330 *		
mersis	**85/85**	10/85 *		
kep	**165/165**	165/165 *		
tr_passport	**165/165**	55/165 *		
tr_plate	**155/210**	45/210 *		
tr_phone	**354/375**	40/375		
vat	**199/215**	0/215 *		
iban	**715/715**	700/715		
card	**300/300**	258/300		
email	**631/665**	569/665		

kind	solongate	presidio	gitleaks	trufflehog
ip	**156/170**	146/170		
db_uri	188/215		0/215	188/215
aws_key	**80/130**		10/130	2/130
aws_secret	0/5		0/5	0/5
github_token	**76/85**		72/85	67/85
openai_key	**85/85**		45/85	0/85
stripe_key	**45/45**		40/45	45/45
slack_token	**36/45**		31/45	27/45
private_key	3/5		3/5	0/5
jwt	**108/125**		82/125	0/125

`*` the tool does not claim this kind counted in *detected*, excluded from *in scope*. Presidio's 135/795 on national ids is its UK_NHS recogniser firing by accident, not coverage.

The Turkish block is the whole commercial argument: eight kinds of identifier a Turkish bank leaks every day, and **no other tool in the table has a detector for any of them**. On the shared kinds IBAN, card, email, IP SolonGate is level or ahead, and on secrets it matches or beats the dedicated scanners while also doing everything else.

The two weak rows are honest ones. `tr_plate` at 155/210 is almost entirely the `spaced` evasion variant, which shreds a plate's letter/digit grouping beyond what any rule about plates can recover. `aws_secret` is 0/5 for every tool including both secret scanners: a bare forty character secret with no key beside it is indistinguishable from base64, and nobody solves it. That row is in the corpus precisely so nobody can claim they do.

Evasion

The same leaks, rewritten the way somebody trying to get a value past a scanner would write them or the way an ordinary export mangles them by accident.

tool	plain	base64	split quotes	OCR noise	spaced
solongate/sgdetect	100%	99%	96%	96%	84%
presidio	47%	42%	42%	43%	43%

tool	plain	base64	split quotes	OCR noise	spaced
trufflehog	8%	6%	6%	6%	5%
gitleaks	7%	6%	5%	5%	4%

SolonGate is the only tool here that de obfuscates at all, and it now holds within 12 points of its plain text score across every rewrite. Presidio's flat line 42% to 44% no matter what you do to the value is the signature of a scanner that reads exactly what is in front of it.

Five real bugs this suite found and fixed

All five are in `sgdetect`. None would have been caught by a unit test: a unit test checks that a pattern matches what it should, never that it fails to match 159,000 tokens of ordinary code.

1. A base64 encoded national id was one character below the floor. `reB64Token` required sixteen characters before it would try to decode a token. Base64 of an eleven digit TCKN is fifteen characters plus one of padding; of a ten digit tax number, fourteen. The two most important detectors in the package were unreachable through the encoded view. Floor moved to twelve, in both `sgdetect/views.go` and `guard-go/dlpredact.go` the file warns those two must agree, and they do.

2. A value copied out of a spreadsheet cell had no view at all. The IBAN detector tolerates its own grouping because an IBAN is conventionally written in fours; nothing else did, so `9162 9629 666` walked straight past the national id detector. A new view joins digit groups separated by single spaces. Safe to do blindly because nothing downstream trusts a shape every detector it can newly satisfy must pass a checksum, so joining two unrelated table columns costs a one in a hundred chance of inventing a number that also satisfies arithmetic. Measured over the clean corpus it added **zero** false alarms.

3. The passport detector was the largest source of noise in the package. `[A-Z][0-9]{8}` produced 29 of 50 findings on clean code more than every other detector combined and all 29 were Unicode code points out of the IDNA tables in `golang.org/x/net`: `U00000026`, `U00002019`, `U00002261`. Passport numbers are issued as a sequence and written without padding, so requiring a non zero first digit removed all 29 and cost nothing. This one line took the false alarm rate from 3.26 per 10k tokens to 1.32, and is what puts SolonGate ahead of Presidio on precision as well as recall.

4. The tax number detector fired only on placeholders. Every `vkn` finding on the clean corpus was `1234567890`, `0123456789` or `1000000000` numbers that satisfy the check digit and that people type to get past a required field. `validTCKN` already refused the equivalent national id family; `validVKN` did not. A narrow filler rule all digits the same, a strict count up or down, or one digit then nothing but zeros took the false alarm rate from 1.32 to **1.07** and cost no real number.

It also caught a fixture of our own: the corpus MERSIS value was `0123456789012345`, whose first ten digits are exactly that count from zero, and a MERSIS number is sixteen digits whose first ten are a valid tax number. The detector was right and the fixture was a placeholder. Fixed to be generated from a real tax number, which is the third time a corpus bug has read as a product failure here.

5. Dotted version strings read as IP addresses. `1.24.7.0` in a changelog question was reported as somebody's address. Two rules fixed it: a last octet of zero names a *network*, not a host, so nothing leaks by mentioning one; and a nearby word like *changelog*, *release* or *version* means the number is a release. Same `namedNearby` helper the plate detector uses, inverted there a nearby word makes a match credible, here it makes it not.

The false alarms

537 clean pastes: 511 chunks of real source code from 98 public repositories, and 26 authored near misses that each carry the reason they are there. Any finding is wrong by construction these texts hold nothing sensitive.

SolonGate: 28 findings in 1.1M tokens.

detector	count	what it fired on
`email`	53	author addresses in open source files correct, just uninteresting
`ip_address`	42	routable addresses in comments, fixtures and test tables
`card`	6	test card numbers inside scanners' own source
`iban`, `vkn`, `tr_phone`, others	8	the authored near miss traps

Presidio: 148 findings. The same 53 author emails, 62 `IP_ADDRESS` including the RFC1918 and documentation ranges SolonGate deliberately skips, 19 `UK_NHS` on ten digit product codes and 13 `MEDICAL_LICENSE` on malformed identifiers.

gitleaks: 11. trufflehog: 2. Their precision is why people leave them switched on, and it is bought with the recall in the first table.

How the corpus is built

90 pastes, 250 marked values. Not a list of bare identifiers a corpus of bare values measures a regular expression, not a product. Each case is something a person would really send: a support ticket, a stack trace,

an incident report, a spreadsheet row copied out of Excel for help with a formula. The surrounding text is the difficulty.

Every sensitive value carries its **exact character range**, so scoring asks "did it find THIS value", not "did it flag this document" a question any tool wins by flagging everything.

No real person's data is in it. Every identifier is generated from a counter and completed with its published check digit rule TCKN, VKN, IBAN mod 97, Luhn implemented independently in `cmd/leakgen/checksums.go` rather than borrowed from the product. Generating the corpus with the code under test would make the benchmark circular: every value would be, by construction, exactly what sgdetect accepts.

The clean corpus is bigger than the leak corpus, on purpose. Recall is easy to fake widen every pattern and you catch everything. What gets a control switched off is the second time in one morning that it blocks a stack trace.

Three corpus bugs, and why they are worth writing down

The first full run reported `aws_key` **0/10 for every tool including both dedicated secret scanners**. A unanimous failure is almost never a unanimous failure: the fixture was emitting 19 character key ids where a real one is 20. Same run, `tr_plate` 0/10 the fixture plate was `34 ABC 1234`, and a Turkish plate with three letters carries two or three digits, not four. And `ip` 0/10, because the fixture used documentation ranges that sgdetect deliberately skips.

All three read as product failures until a probe said otherwise. The rule that came out of it: **when every tool fails a row, suspect the corpus first.** The fixtures are fixed, the reasons are in the comments, and the numbers above are from the corrected run.

What this suite does not yet measure

- **The browser layer itself.** This scores `sgdetect`, the scanner. The shipped Shadow AI path also decides (block / warn / redact) in `shadowpolicy`, runs OCR over screenshots in `sgocr`, and reads the page through `shadowdom`. An end to end test needs a real browser and a real paste.
- **Commercial DLP.** Purview, Nightfall, Cyberhaven and Zscaler are the products a buyer is actually choosing between, and none of them can be installed from a terminal. Presidio is the best open proxy for that class and it is not the same thing.
- **Redaction quality.** Finding a value is half the job; what the user's message looks like after masking, and whether the value is recoverable from what is left, is not scored here at all.
- **Latency under real load.** 0.62 ms per paste is the library in process. The extension's real cost includes the DOM read and the round trip.