SUNIMOD FIELD NOTES

The API Key You Forgot: How Hardcoded Secrets Become a Business Breach.

A single API key committed to source code can expose customer data, create fraudulent cloud costs, interrupt operations, and give an intruder a path into connected systems. Learn how the weakness works, reproduce it safely in a local lab, and fix it at the code, hosting, and process levels.

BY Christopher Jacobson

PUBLISHED July 15, 2026

TOPIC CISSP, Domain 1

https://sunimod.com/the-api-key-you-forgot-how-hardcoded-secrets-become-a-business-breach/

The central lesson An API key is not ordinary configuration. It is delegated authority. When the key is copied into source code, the authority travels with every clone, backup, build artifact, support archive, and old commit that contains it.

What you will learn

1. What the weakness is
2. How exploitation works
3. A vulnerable WordPress example
4. A safe localhost lab
5. Where secrets escape
6. How to detect exposure
7. Potential repercussions
8. What to do after a leak
9. How to design the fix
10. A solvable business service

1. What is the weakness?

The weakness is **hardcoded credentials**: a password, API token, private key, database connection secret, webhook signing secret, or other authenticator is written directly into application code or a file that travels with the application. Some values identify an account but do not authenticate it. A public client identifier may be safe to publish. A secret token is different because a service accepts it as proof that the request is authorized. With a bearer token, possession is often enough: the receiving service does not know whether the request came from your production application or from someone who copied the token.

Value	Typical purpose	May it be public?
Account ID or public client ID	Identifies an account or application	Sometimes; verify the provider's design
API secret or bearer token	Authorizes API operations	**No**
Webhook signing secret	Proves that an incoming event is authentic	**No**
Database password	Authenticates to a database	**No**
Private key	Signs, decrypts, or authenticates	**No**; only the matching public key is public

A private repository lowers exposure; it does not turn source control into a secret manager. Developers, contractors, automation, integrations, backups, compromised accounts, and accidental visibility changes may still expose the repository. The safer design keeps reusable credentials out of the codebase.

2. How exploitation works, step by step

This vulnerability usually becomes an incident through a chain of ordinary events. No advanced memory corruption exploit is required. The attacker abuses valid authority that the business accidentally disclosed.

1. **A credential enters the codebase.** A developer needs an integration to work quickly and pastes a token into PHP, JavaScript, a configuration file, a test fixture, or a deployment script.
2. **The credential is copied.** Git stores the commit. CI systems clone it. Developers make local copies. Backups, staging servers, container layers, and support archives may preserve it.
3. **An unauthorized party discovers it.** Discovery can be manual or automated. Secret scanners look for known token formats, suspicious variable names, high-entropy strings, and credentials that can be validated with the issuing provider.
4. **The party tests the credential.** A low-impact request may reveal whether the token is live, what account owns it, and what permissions it has.
5. **The valid credential is used as intended—but by the wrong person.** The attacker may read data, create resources, alter settings, send messages, retrieve backups, modify DNS, publish software, or create additional credentials, depending on the granted permissions.
6. **The original leak becomes difficult to contain.** Removing the visible line does not revoke the credential, erase clones, invalidate old commits, or undo actions already performed.

The size of the incident is controlled by four factors: the credential's permissions, how long it remains valid, which systems accept it, and how quickly monitoring detects abnormal use. A read-only token limited to one dataset is not harmless, but it has a smaller blast radius than an administrator token shared across production, staging, backups, and deployment automation.

3. An intentionally vulnerable WordPress/PHP example

The following function calls an external order service. The token is fictitious and the `.invalid` domain is intentionally non-routable. The defect is the location of the credential: it is embedded in source code.

PHP · intentionally vulnerable

```php
<?php
declare(strict_types=1);

if (!defined('ABSPATH')) {
    exit;
}

function sunimod_demo_fetch_orders_vulnerable() {
    $api_token = 'SUNIMOD_DEMO_TOKEN_2026_DO_NOT_USE';

    $response = wp_remote_get(
        'https://billing-api.example.invalid/v1/orders',
        [
            'timeout' => 15,
            'headers' => [
                'Accept' => 'application/json',
                'Authorization' => 'Bearer ' . $api_token,
            ],
        ]
```

```
    );

    if (is_wp_error($response)) {
        return $response;
    }

    return json_decode(
        wp_remote_retrieve_body($response),
        true
    );
}
```

The application may work perfectly. That is what makes this defect easy to miss during functional testing. The request succeeds because the service trusts the token. Unfortunately, anyone who obtains the source can copy the same string and construct the same authorization header.

Why changing the variable name does not help

Calling the value `$configuration`, encoding it with Base64, splitting it across two strings, or placing it in a compiled bundle does not remove the secret. Those techniques obscure the value; they do not establish an access-control boundary. The running application must reconstruct the credential, so a party with sufficient access to the code or runtime can usually reconstruct it too.

Why front-end JavaScript is especially dangerous

Code sent to a browser is delivered to the user's device. Minification and bundling do not make a server-side secret confidential. A privileged API token placed in browser JavaScript should be treated as publicly disclosed. Browser applications normally authenticate users and call a controlled backend, while the backend holds narrowly scoped server credentials or uses delegated, short-lived tokens.

4. Safe localhost lab: possession becomes access

> **Use this lab only on your own computer.** It listens on `127.0.0.1`, uses a dummy token, returns fabricated orders, and does not contact a third-party system. Do not test credentials or repositories without explicit authorization.

Save the following file as `mock-api.php`. It represents a small API that trusts one bearer token. The comparison uses `hash_equals()` to avoid an avoidable timing-sensitive string comparison, but the central risk remains: whoever possesses the accepted token receives the protected response.

PHP · mock-api.php

```
<?php
declare(strict_types=1);

const EXPECTED_TOKEN = 'SUNIMOD_DEMO_TOKEN_2026_DO_NOT_USE';

header('Content-Type: application/json; charset=utf-8');

$method = $_SERVER['REQUEST_METHOD'] ?? 'GET';
$request_uri = $_SERVER['REQUEST_URI'] ?? '/';
$path = parse_url($request_uri, PHP_URL_PATH);
$authorization = $_SERVER['HTTP_AUTHORIZATION'] ?? '';

if ($method !== 'GET' || $path !== '/v1/orders') {
    http_response_code(404);
```

```
    echo json_encode(
        ['error' => 'not_found'],
        JSON_PRETTY_PRINT | JSON_UNESCAPED_SLASHES
    );
    exit;
}

$expected_header = 'Bearer ' . EXPECTED_TOKEN;

if (!hash_equals($expected_header, $authorization)) {
    http_response_code(401);
    header('WWW-Authenticate: Bearer');
    echo json_encode(
        ['error' => 'invalid_or_missing_token'],
        JSON_PRETTY_PRINT | JSON_UNESCAPED_SLASHES
    );
    exit;
}

http_response_code(200);
echo json_encode(
    [
        'orders' => [
            [
                'order_id' => 'DEMO-1042',
                'customer' => 'Northwind Demo Company',
                'total' => 249.00,
                'status' => 'processing',
            ],
            [
                'order_id' => 'DEMO-1043',
                'customer' => 'Contoso Training Account',
                'total' => 89.50,
                'status' => 'paid',
            ],
        ],
    ],
    JSON_PRETTY_PRINT | JSON_UNESCAPED_SLASHES
);
```

Run the server and make both an unauthorized and an authorized request:

Bash · Ubuntu localhost lab

```
# Ubuntu: install the PHP command-line runtime used by this lab.
sudo apt update
sudo apt install php-cli curl

# Save the PHP code above as mock-api.php, then start the local API.
php -S 127.0.0.1:8081 mock-api.php

# In a second terminal, make a request without the leaked token.
curl -i http://127.0.0.1:8081/v1/orders

# Now present the hardcoded token found in the source code.
curl -i \
  -H 'Authorization: Bearer SUNIMOD_DEMO_TOKEN_2026_DO_NOT_USE' \
  http://127.0.0.1:8081/v1/orders
```

The first request receives `401 Unauthorized`. The second request receives the fabricated order data because it presents the expected token. The mock API cannot distinguish the legitimate application from a person who copied the credential. That is the essence of the exploit.

> **The token is the authorization.** Encrypting the network connection with HTTPS protects the token while it travels, but HTTPS cannot protect a token that was already disclosed in source code, a build log, a backup, or a repository.

5. Where hardcoded secrets escape

Teams often imagine one exposure path: “the repository became public.” In practice, the same credential may exist in many places, and every copy extends the response effort.

Exposure path	How the secret survives	Why the risk is missed
Git history	The original commit remains reachable after a later deletion	The current branch looks clean
CI/CD logs	Commands, environment dumps, debug output, or failed jobs print it	Build logs are treated as operational noise
Build artifacts	Compiled bundles, ZIP files, source maps, and container layers include it	The source repository itself may be private
Backups and staging	Old site copies preserve old configuration and credentials	Non-production systems receive less monitoring
Support and chat	A developer pastes configuration into a ticket or message	The tool is trusted for collaboration, not secret custody
Browser code and source maps	The secret is delivered to every visitor	Minification is mistaken for confidentiality
Former vendors or employees	Long-lived shared credentials remain valid after access should end	The credential has no clear owner or rotation event

6. How to detect exposed credentials

Detection should cover both the current working tree and historical commits. A simple keyword search is a useful first pass, but it will miss secrets with unfamiliar names and produce false positives. Dedicated scanners add provider-specific patterns, entropy checks, allowlists, and history-aware analysis.

Bash · authorized repository review

```
# Search tracked files in the repository you are authorized to inspect.
git grep -nEi \
  '(api[_-]?key|secret|token|password)[[:space:]]*[:=]'

# Search patch history, because deleting a line in a later commit
# does not remove the earlier committed value.
git log --all -p | grep -nEi \
  '(api[_-]?key|secret|token|password)[[:space:]]*[:=]'

# Run a dedicated scanner and redact detected values from terminal output.
gitleaks git --redact --verbose .
```

Treat scanner output as sensitive. Reports may contain file paths, commit metadata, and—unless redacted—the credentials themselves. Store findings with restricted access and avoid pasting live values into ordinary tickets.

What a scanner cannot prove

A clean scan does not prove that no secret exists. Custom token formats, encrypted archives, screenshots, database rows, binary files, runtime-only values, and secrets split across variables may evade pattern-based detection. Combine automated scanning with architecture review, code review, access-log analysis, and an inventory of every integration that requires authentication.

Prevent the commit instead of only detecting it later

Repository push protection and pre-commit scanning can interrupt the mistake before the credential reaches shared history. Prevention is cheaper because incident response may require rotation, application changes, log review, collaborator coordination, and history rewriting.

YAML · .pre-commit-config.yaml

```yaml
repos:
  - repo: https://github.com/gitleaks/gitleaks
    rev: v8.30.1
    hooks:
      - id: gitleaks
```

Bash · enable and test the hook

```bash
# Run from the repository root after pre-commit and Gitleaks are installed.
pre-commit install
pre-commit run --all-files
```

7. The permissions decide the blast radius

A leaked credential does not grant “access” in the abstract; it grants the exact actions permitted by the receiving system. The most important design question is therefore not merely “Where is the token stored?” but also “What can this token do?”

- Limit the credential to one application, one environment, and the minimum required operations.
- Separate read, write, administration, deployment, backup, and billing authority where the provider permits it.
- Prefer short-lived credentials or workload identity over reusable static secrets.
- Restrict source networks, audiences, resources, or API scopes when supported.
- Give every credential an owner, purpose, creation date, rotation path, and emergency revocation procedure.
- Log use by credential or workload identity so abnormal activity can be investigated quickly.

Least privilege converts a potentially company-wide incident into a smaller, more understandable event. It does not excuse a leak, but it limits how much authority can be stolen and makes abnormal behavior easier to distinguish from normal application activity.

8. Potential repercussions for your business or infrastructure

The technical event is “an unauthorized party used a valid credential.” The business consequences depend on the connected service, the data it can reach, and the actions it can perform.

Repercussion	What may happen	Business effect
Customer data exposure	Orders, profiles, documents, addresses, or support records are retrieved	Notification work, legal review, customer harm, lost trust, and contract issues
Data tampering	Records, prices, account details, permissions, or configurations are changed	Incorrect decisions, fraud, reconciliation cost, and uncertainty about data integrity
Cloud cost abuse	Compute, storage, messaging, AI, or other metered resources are created or consumed	Unexpected invoices, quota exhaustion, and emergency shutdowns
Service interruption	Resources are deleted, rate limits are exhausted, or the provider suspends the account	Lost sales, missed work, support volume, and recovery time
Fraudulent communication	SMS, push notifications, support messages, or transactional messages are sent through your account	Direct fees, customer deception, brand damage, and provider enforcement
Software supply-chain compromise	A token with deployment or package-publishing rights alters released software	Downstream customer compromise, emergency releases, and severe trust loss
Persistent access	The intruder creates users, keys, webhooks, scheduled jobs, or alternative access paths	Rotating the original token does not fully remove the intruder
Backup compromise	Backups are read, deleted, encrypted, or made unreliable	Recovery options shrink exactly when the business needs them most
Trade-secret loss	Source code, pricing, internal documents, or proprietary data are copied	Competitive harm and difficult-to-measure long-term loss
Incident-response expense	Teams rotate credentials, review logs, rebuild systems, contact vendors, and validate data	Unplanned labor, consulting expense, delayed projects, and management distraction

> **Shared credentials magnify uncertainty.** When production, staging, contractors, and multiple applications use the same token, logs may show what the token did without showing which legitimate workload normally used it. Unique credentials improve both containment and investigation.

9. What to do when a secret has leaked

The safest working assumption is that a disclosed credential may already be compromised. Do not wait for obvious damage before containing it, especially when the credential is public, broadly shared, highly privileged, or long-lived.

1. **Revoke or rotate the credential first.** Removing the line from code is not containment. Create a replacement through the provider’s approved process, update legitimate workloads, disable the old credential, confirm service health, and then delete the old credential when appropriate.

2. **Preserve evidence.** Record the time discovered, repository, file, commit, credential owner, provider, affected environment, and actions taken. Preserve relevant logs before retention periods expire.
3. **Determine the authority and exposure window.** Identify permissions, resources, creation date, last use, all systems that stored the secret, and the earliest time an unauthorized party could have obtained it.
4. **Review provider and application logs.** Look for unfamiliar source addresses, impossible timing, unusual regions, new identities, permission changes, exports, deletions, high-volume requests, billing spikes, and activity outside the application's normal pattern.
5. **Hunt for persistence and secondary impact.** Check for new credentials, users, roles, webhooks, scheduled tasks, deployment changes, altered recovery settings, modified backups, and secrets retrieved using the original credential.
6. **Remove the secret from current code and, when justified, repository history.** Coordinate history rewriting carefully. Clones and forks may retain the old data, which is another reason revocation comes first.
7. **Assess notification and contractual duties.** Involve the appropriate security, leadership, privacy, legal, insurance, customer, and vendor contacts based on the evidence and the data or services affected.
8. **Correct the system that allowed the leak.** Add a secure runtime retrieval method, narrow permissions, separate credentials by environment, enable secret scanning, improve logging, and document ownership and rotation.

> **Order matters:** invalidate the authority, preserve evidence, investigate use, remove residual copies, and then improve the design. A clean Git diff alone is not a completed response.

10. How to design the fix

The goal is not to move the same long-lived secret from one obvious file to a slightly less obvious file. A durable fix combines secure storage, runtime delivery, minimal permissions, rotation, monitoring, and clear ownership.

A practical WordPress pattern for smaller environments

For a conventional WordPress deployment, the hosting environment can supply the token outside the repository and outside the public document root. The following `wp-config.php` fragment fails closed when the value is absent. The environment and server configuration still require strict access controls; an environment variable is a delivery mechanism, not a vault.

PHP · wp-config.php

```php
<?php
declare(strict_types=1);

/*
 * Add this before WordPress loads wp-settings.php.
 * The hosting environment must provide SUNIMOD_CRM_API_TOKEN
 * outside the repository and outside the public web directory.
 */
```

```
$sunimod_crm_token = getenv('SUNIMOD_CRM_API_TOKEN');

if (is_string($sunimod_crm_token) && trim($sunimod_crm_token) !== '') {
    define('SUNIMOD_CRM_API_TOKEN', trim($sunimod_crm_token));
} else {
    error_log('SUNIMOD_CRM_API_TOKEN is not configured.');
}

unset($sunimod_crm_token);
```

The integration code reads the defined value, rejects a missing credential, prevents redirects, verifies the HTTP status, and validates the returned JSON before using it.

PHP · server-side integration

```
<?php
declare(strict_types=1);

if (!defined('ABSPATH')) {
    exit;
}

function sunimod_fetch_orders_securely() {
    if (
        !defined('SUNIMOD_CRM_API_TOKEN') ||
        !is_string(SUNIMOD_CRM_API_TOKEN) ||
        SUNIMOD_CRM_API_TOKEN === ''
    ) {
        return new WP_Error(
            'sunimod_missing_crm_token',
            'The CRM integration credential is not configured.'
        );
    }

    $response = wp_remote_get(
        'https://billing-api.example.invalid/v1/orders',
        [
            'timeout' => 15,
            'redirection' => 0,
            'headers' => [
                'Accept' => 'application/json',
                'Authorization' => 'Bearer ' . SUNIMOD_CRM_API_TOKEN,
            ],
        ]
    );

    if (is_wp_error($response)) {
        return $response;
    }

    $status_code = wp_remote_retrieve_response_code($response);

    if ($status_code < 200 || $status_code >= 300) {
        return new WP_Error(
            'sunimod_crm_http_error',
            'The CRM service returned an unexpected response.',
            ['status_code' => $status_code]
        );
    }

    $decoded_body = json_decode(
        wp_remote_retrieve_body($response),
        true
    );

    if (!is_array($decoded_body)) {
        return new WP_Error(
            'sunimod_crm_invalid_json',
            'The CRM service returned invalid JSON.'
        );
```

```
    }

    return $decoded_body;
}
```

In higher-risk or cloud-native environments, prefer a dedicated secret-management service and workload identity so the application can retrieve a narrowly scoped secret at runtime—or avoid a static secret entirely. Short lifetimes and automatic rotation reduce the time a copied credential remains useful.

Keep local secret files out of Git

A `.gitignore` rule prevents new untracked files from being added accidentally. It does not remove a file already committed, and it does not protect the file from other local users, backups, malware, or web server exposure.

Git · .gitignore

```
# Local runtime configuration
.env
.env.*
!.env.example

# Credentials and private key material
auth.json
credentials.json
secrets/
*.pem
*.p12

# Local overrides
wp-config-local.php
config.local.php
```

Block hidden configuration files at the web server

Web-server denial rules are defense in depth. Secrets still belong outside the document root, and a rule must be tested against the site's actual routing and certificate-renewal configuration.

Nginx · Ubuntu server

```
# Nginx defense in depth: deny hidden files except ACME challenges.
location ~ /\.(?!well-known(?:/|$)) {
    deny all;
    access_log off;
    log_not_found off;
}
```

Apache 2.4 · Ubuntu server

```
# Apache 2.4 defense in depth: deny hidden files.
<FilesMatch "^\.">
    Require all denied
</FilesMatch>
```

Build a credential lifecycle, not a one-time cleanup

- Inventory each secret, owner, purpose, provider, environment, permissions, storage location, and consuming workload.

- Use separate credentials for development, staging, and production.
- Rotate on exposure, personnel or vendor changes, privilege changes, and provider-recommended schedules.
- Test rotation before an emergency so replacement does not create an avoidable outage.
- Alert on unusual use, denied requests, new source locations, privilege changes, and unexpected spend.
- Prohibit secrets in browser code, source maps, examples, screenshots, tickets, and ordinary documentation.
- Review third-party plugins and integrations that request broad, permanent tokens.

11. Turn the risk into a solvable business service

This problem is well suited to a focused **Secrets Exposure and Credential Hardening Review**. The engagement connects technical findings to the systems, customers, revenue, and recovery obligations that matter to the organization.

What the engagement can include

- Authorized scanning of application code, Git history, deployment files, and selected build artifacts.
- An inventory that maps credentials to owners, environments, permissions, integrations, and business functions.
- Risk ranking based on exposure, privilege, data access, credential lifetime, and available monitoring.
- Immediate rotation and containment planning for confirmed live secrets.
- WordPress, web application, hosting, and integration changes that remove hardcoded credentials.
- Least-privilege redesign and separation of production, staging, development, vendor, and automation access.
- Secret scanning in local development and CI/CD, plus repository push protection where available.
- Rotation runbooks, ownership records, recovery steps, and an incident-ready evidence checklist.
- A prioritized remediation roadmap that distinguishes urgent containment from longer-term modernization.

The result is more than a scanner report. It is a practical plan for removing exposed authority, keeping the application working, reducing the blast radius of future mistakes, and making ownership clear.

Concerned that a website, custom application, repository, or integration may contain exposed credentials?

Hire **Sunimod** to evaluate the application and its surrounding workflow, identify practical remediation steps, and build the safer implementation. Do not send passwords, API keys, access tokens, or other secrets through the form; Sunimod can arrange a safer handoff when access is required. Request a Sunimod project quote

Good reasons to start now

- You inherited a website or application and do not know how its integrations are authenticated.
- A repository, ZIP archive, backup, or staging environment was shared outside the intended team.
- A developer, employee, contractor, or agency with access is leaving.
- You found a token in code, an old commit, browser JavaScript, a log, or a support ticket.
- Your cloud or API bill changed unexpectedly, or an integration shows unfamiliar activity.
- You are modernizing a WordPress site or custom application and want safer access designed into the work.

12. Frequently asked questions

Is a `.env` file automatically secure?

No. It is safer than committing the value directly only when the file remains outside version control, is not web-accessible, has restrictive permissions, is excluded from logs and backups that do not need it, and is delivered securely. A managed secret store or workload identity may provide stronger access control, auditing, and rotation.

Is deleting the secret from the latest version enough?

No. The credential may remain valid, and earlier commits, clones, forks, build artifacts, backups, or messages may still contain it. Revoke or rotate first, then address residual copies and history based on the exposure.

Does a private repository solve the problem?

It reduces public exposure but does not provide the controls of a dedicated secrets system. Private repositories still have users, automation, integrations, backups, access tokens, and the possibility of account compromise or accidental visibility changes.

Can secret scanning guarantee that the codebase is clean?

No. Scanning is an important layer, not proof of absence. Review architecture, runtime configuration, browser bundles, logs, backups, third-party services, binary artifacts, and the inventory of expected credentials.

Should every secret rotate on the same schedule?

No. Rotation should reflect the provider, credential type, exposure, permissions, operational impact, and applicable requirements. Every organization should be able to rotate immediately after suspected exposure, even when routine rotation intervals differ.

Key takeaways

- A secret in source code is authority copied into every place the code travels.
- An attacker may not need to “break” the API; presenting a valid leaked token can be enough.
- Deleting the visible line does not revoke the credential or remove it from history.
- Containment starts with revocation or rotation, followed by evidence preservation and investigation.
- Secure storage must be paired with least privilege, short lifetimes, monitoring, ownership, and tested rotation.
- Pre-commit scanning and push protection reduce the chance that the mistake reaches shared history.

Sources and further reading

1. MITRE CWE-798: Use of Hard-coded Credentials
2. OWASP Secrets Management Cheat Sheet
3. IETF RFC 6750: OAuth 2.0 Bearer Token Usage
4. GitHub Docs: Push protection
5. GitHub Docs: Remediating a leaked secret
6. GitHub Docs: Removing sensitive data from a repository
7. AWS Prescriptive Guidance: Prevent application secrets from being exposed
8. NIST CSRC Glossary: Least privilege
9. Gitleaks: Git repository secret scanner

This article is educational and is not legal advice. Test only systems and repositories you own or are explicitly authorized to assess. Incident response, notification, and evidence-handling decisions should be adapted to the facts, contracts, insurance conditions, provider requirements, and applicable law.