
COMMON SECURITY
PITFALLS
A Software Engineer's Perspective
Lochie Ashcroft

$ whoami

Computer Science

- Strong fundamentals
- Did well in CTFs during uni
- Prefer designing (defence) over attacking

IBM Security - R&D Lab

- Identity and Access Management (Verify SaaS)
- Software Engineer (Intern → Mid)

Stake - Investment Platform

- Software Engineer (Product, Platform)
- Site Reliability Engineer

What this talk is not

ATT&CK Matrix for Enterprise

layout: flat | show sub-techniques | hide sub-techniques

The Ten Most Critical Web Application Security Risks

Introduction

Welcome to the 8th installment of the OWASP Top Ten!

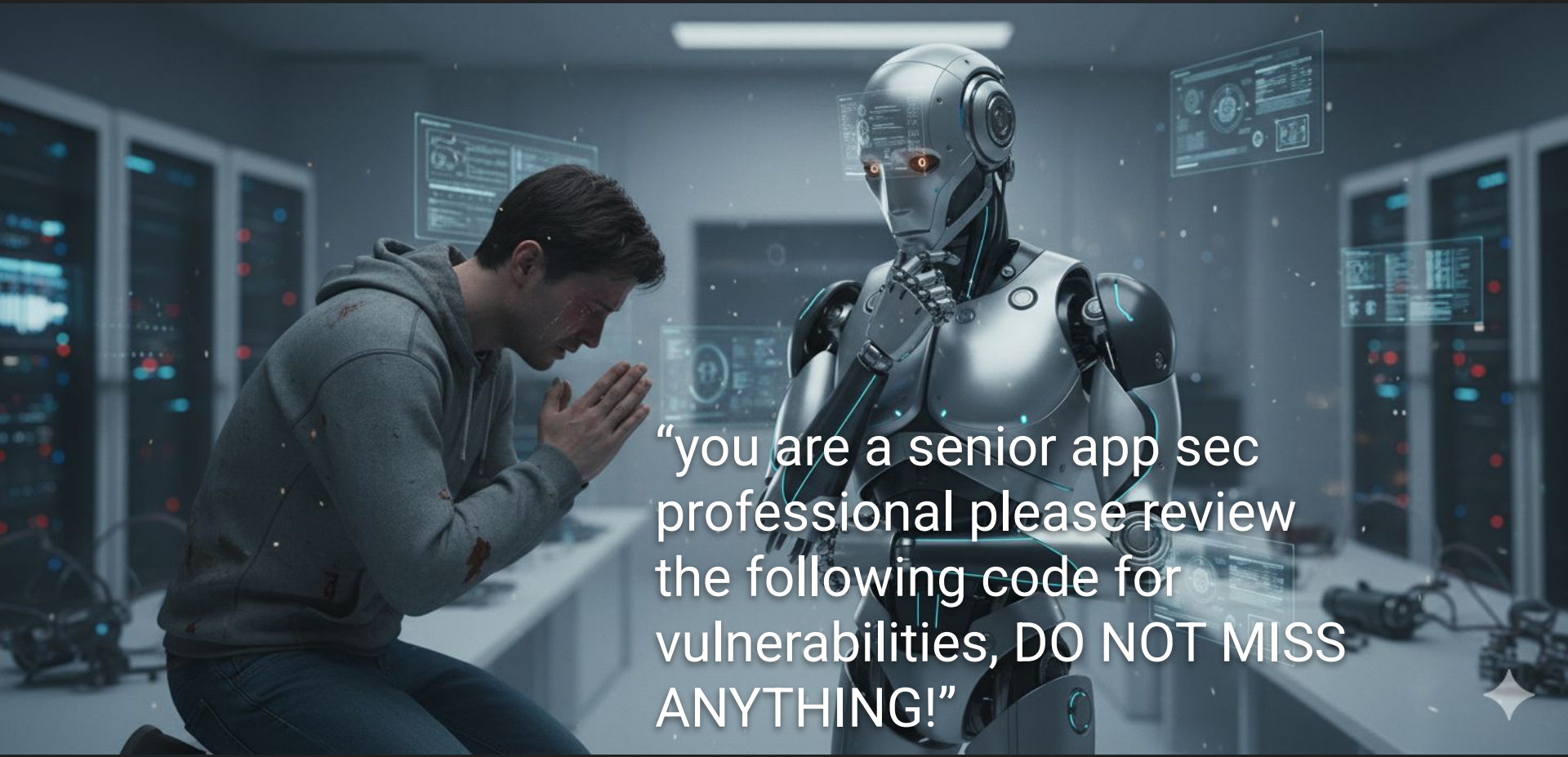
A huge thank you to everyone who contributed data and perspectives in the survey. Without you, this installment would not have been possible. **THANK YOU!**

Introducing the OWASP Top 10:2025

- A01:2025 - Broken Access Control
- A02:2025 - Security Misconfiguration
- A03:2025 - Software Supply Chain Failures
- A04:2025 - Cryptographic Failures
- A05:2025 - Injection
- A06:2025 - Insecure Design
- A07:2025 - Authentication Failures
- A08:2025 - Software or Data Integrity Failures
- A09:2025 - Security Logging & Alerting Failures
- A10:2025 - Mishandling of Exceptional Conditions

AI

?


“you are a senior app sec professional please review the following code for vulnerabilities, DO NOT MISS ANYTHING!”

What this talk is

Types of Engineers

Throughput Optimizer

Metric: Ticket velocity and sprint completion.

Philosophy: "Done is better than perfect."

Constraint: Avoiding blockers and minimizing scope creep.

Sustainability Optimizer

Metric: System uptime and long-term maintainability.

Philosophy: "Build it once, build it right."

Constraint: Balancing technical debt against feature requests.

Corporate Reality

Company Stage	Primary Driver	Security Posture
Early Stage	Survival & Product-Market Fit	"Security is Friction" Speed to market is the only metric that ensures the next round of funding.
Growth	Market Capture	"The Complexity Gap" Scaling features faster than the underlying security infrastructure can support.
Mature / Public	Risk & Compliance	"Defensive Investment" Security is often driven by audit requirements and legal liability rather than proactive engineering.

The Incentive Gap

Invisible Work

Feature wins are celebrated in All-Hands meetings; "the breach that never happened" is rarely noticed.

Asymmetric Risk

The individual developer often bears the "speed pressure", while the company bears the "security risk".

The Cost of Friction

If security tools add 20% more time to a deployment, the "Throughput Optimizer" will naturally find workarounds.


“If you fail to plan, you are planning to fail”
Benjamin Franklin

Design Document

- What's the problem
- Potential Solutions
- Assumptions
- Constraints
- Architecture
- Flows
- Security
- …

Security - Threat Model

What you are protecting from whom

- Assets
- Attackers
- Entry Points
- Trust Boundaries

Security - Requirements

Determined by business and regulatory needs

- Authentication and authorization standards
- Data encryption in transit and at rest
- Logging and monitoring requirements
- Secure error handling


DEEESIVE PROGRAMMING
INPUT SANITIZED!
TRY/SANITZED)
TRY/CATCH
TRY/CATCH
TRY/SATCH
TRY/CATCH
TRY/CATCH
TRY/SATTZER)
TRY/CANITZER)
TUR SANITIZED!
NULL CHECK PASSED!
DEFENSIVE PROGRAMMING
THIR SANITIZED!
L WORINCE

Defensive Programming - Super Simple Example

```
1
2  def divide(x, y):
3      return x / y
```

Defensive Programming - Solid Divide

```python
from typing import Union

Number = Union[int, float]


def divide(x: Number, y: Number) -> float:
    """Divide two numbers.

    Args:
        x (Number): the numerator
        y (Number): the denominator

    Raises:
        TypeError: if either input is not a number
        ValueError: if the denominator is zero

    Returns:
        float: the result of the division
    """
    if not isinstance(x, (int, float)) or not isinstance(y, (int, float)):
        raise TypeError("Both inputs must be numbers (int or float).")

    if y == 0:
        raise ValueError("Denominator cannot be zero.")

    return x / y
```

Defensive Programming - Running

```
28
29  a, b = 10, 2
30  print(divide(a, b))
31
32  c, d = "10", 2
33  print(divide(c, d))
34
```

```
1  $ python3 defense-programming.py
2  5.0
3  Traceback (most recent call last):
4    File "/SOME/PATH/script.py", line 33, in <module>
5      print(divide(c, d))
6            ^^^^^^^^^^^^
7    File "/SOME/PATH/script.py", line 21, in divide
8      raise TypeError("Both inputs must be numbers (int or float).")
9  TypeError: Both inputs must be numbers (int or float).
10
```

Defensive Programming - Static Type Analysis

```
$ mypy script.py
script.py:33: error: Argument 1 to "divide" has incompatible type "str"; expected "int | float"  [arg-type]
Found 1 error in 1 file (checked 1 source file)
```

: my[py]

https://github.com/python/mypy

> write code...
> git add script.py
> git commit -m > “implement divide”
> git push
> deploy to prod
> eat lunch

> write code
> how could it break?
> think...
> write tests...
> write tests...
> run tests
> git add script.py
> git commit -m > “impl divide”
> git push
> review
> deploy to prod

```python
class DivideTestCase(unittest.TestCase):
    def test_positive_numbers(self):
        self.assertEqual(divide(10, 2), 5)
        self.assertEqual(divide(3.14, 1.57), 2.0)

    def test_negative_numbers(self):
        self.assertEqual(divide(-10, 2), -5)
        self.assertEqual(divide(10, -2), -5)

    def test_decimal_results(self):
        self.assertAlmostEqual(divide(5, 2), 2.5)
        self.assertAlmostEqual(divide(1, 3), 0.333333, places=6)

```

```
    def test_division_by_zero(self):
        with self.assertRaises(ValueError):
            divide(10, 0)

    def test_non_numeric_inputs(self):
        with self.assertRaises(TypeError):
            divide("10", 2)
        with self.assertRaises(TypeError):
            divide(None, 2)
        with self.assertRaises(TypeError):
            divide([1, 2, 3], 2)
        with self.assertRaises(TypeError):
            divide({"a": 1, "b": 2}, 2)

    def test_large_numbers(self):
        expected_result = 1e50
        actual_result = divide(1e100, 1e50)
        self.assertAlmostEqual(actual_result, expected_result, delta=1e34)

    def test_small_numbers(self):
        self.assertAlmostEqual(divide(1e-100, 1e-50), 1e-50)

    def test_boundary_cases(self):
        self.assertAlmostEqual(divide(1.79e308, 1), 1.79e308)
        self.assertAlmostEqual(divide(2.23e-308, 1), 2.23e-308)
```

Defenseless Programming - Happy Path Only

- Didn’t consider what could go wrong
- No input validation
- No response validation
- Limited error handling
- Lack of Authorization
- Tests ❓

- LGTM! 🤙🚀

Input Validation

Defenseless Programming - Input Validation - SQLi

```sql
-- Bad: concatenating user input directly
SELECT * FROM users
WHERE username = 'admin' AND password = 'password';

-- Attacker supplies:
-- username = admin
-- password = ' OR '1'='1

-- Resulting query (insecure string concatenation):
SELECT * FROM users
WHERE username = 'admin' AND password = '' OR '1'='1';
```

Defenseless Programming - Input Validation - SQLi

```
// Node.js (vulnerable) - naive builder that concatenates user input
app.get('/users', (req, res) => {
  // optional query params: ?name=&age=&role=
  const { name, age, role } = req.query;

  // Start with always-true base so we can append "AND ..." or "OR ..." easily
  let sql = "SELECT * FROM users WHERE 1=1";

  if (name) sql += " OR name = '" + name + "'";
  if (age)  sql += " OR age = " + age;
  if (role) sql += " OR role = '" + role + "'";

  // run raw query (vulnerable)
  db.query(sql).then(rows => res.json(rows));
});
```

Defenseless Programming - Timing Attacks

```
app.post("/login", async (req, res) => {
  const { email, password } = req.body;

  const user = await db.users.findOne({ email: email });

  if (!user) {
    return res.status(401).json({
      error: "Invalid email or password"
    });
  }

  const passwordMatch = await bcrypt.compare(password, user.passwordHash);

  if (!passwordMatch) {
    return res.status(401).json({
      error: "Invalid email or password"
    });
  }

  res.json({ message: "Login successful" });
});
```

Defenseless Programming - Timing Attacks

```
const DUMMY_HASH = "$2b$10$KbQiY2h3p2q2S3N6r5uQOe8vKZcQ1qKjY8xj2mJq1b3sZ4Kp6dN9a";

app.post("/login", async (req, res) => {
  const { email, password } = req.body;

  const user = await db.users.findOne({ email: email });

  const hash = user ? user.passwordHash : DUMMY_HASH;

  const passwordMatch = await bcrypt.compare(password, hash);

  if (!user || !passwordMatch) {
    return res.status(401).json({
      error: "Invalid email or password"
    });
  }

  res.json({ message: "Login successful" });
});
```

Defenseless Programming - Input Validation - Directory Traversal

Endpoint: foo.com/view?file=**{filename}**

Defenseless Programming - Input Validation - Directory Traversal

Normal Request

curl "foo.com/view?file=report.txt"

Malicious Request

curl "foo.com/view?file=../../../../etc/passwd"

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
...
```

Defenseless Programming - Input Validation - Encoding

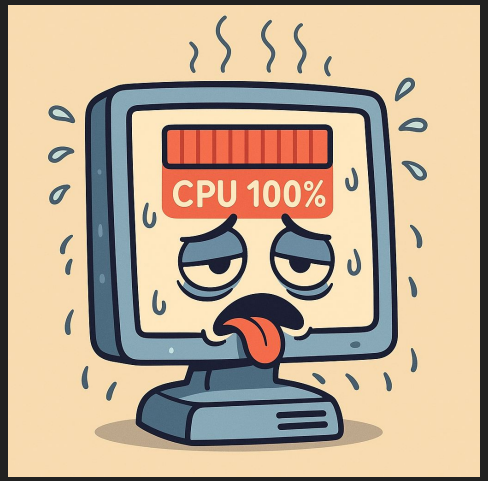
Character	URL Encoded	Double Encoded	Common Use in Attacks
/	%2F	%252F	Path traversal
.	%2E	%252E	Combined with /
?	%3F	%253F	Query string manipulation
&	%26	%2526	Fake query params
%	%23	%2525	Enables double encoding
<	%3C	%253C	Script tags
>	%3E	%253E	Script Tags
;	%3B	%253B	Command chaining, shell injection

Defenseless Programming - Input Validation - Encoding

Scientific Notation

1e2 = 1×10^{2} = 100

1e100000 = 1×10^{100000} = 1000000000000000000000000000000000000000000000...



Defenseless Programming - Input Validation - Java BigDecimal

```
public static class InputData {
    public BigDecimal value;

    @Override
    public String toString() {
        return "InputData{value=" + value + '}';
    }
}
```

Defenseless Programming - Input Validation - Java BigDecimal

```
ObjectMapper mapper = new ObjectMapper();
InputData data = mapper.readValue(jsonString, InputData.class);
```

Defenseless Programming - Input Validation - Java BigDecimal

```
...
int digits = 1000000;

StringBuilder sb = new StringBuilder(digits + 16);
sb.append("{\"value\": ");

for (int i = 0; i < digits; i++)
    sb.append('9');

sb.append("}");

String chosen = sb.toString();
...
```

Defenseless Programming - Input Validation - Java BigDecimal

```
Duration: 8585 MS
Memory  : 226  MB
```

Defenseless Programming - Input Validation - Java BigDecimal

```
com.fasterxml.jackson.core.exc.StreamConstraintsException -
Number length (1000000) exceeds the maximum length (1000)
```

Patched: v2.15.0 (April 2023)

Defenseless Programming - Input Validation - Java BigDecimal

Other Mitigations

- Create custom deserialization logic to validate before mapping
- Use JSON schemas to validate data shape before mapping
- Use appropriate data types for business logic
- Limit request payload size

Defenseless Programming - Input Validation - Regex

Regex Pattern: ^(a+)+$

^ – Start of the string

(a+) – One or more a characters, grouped

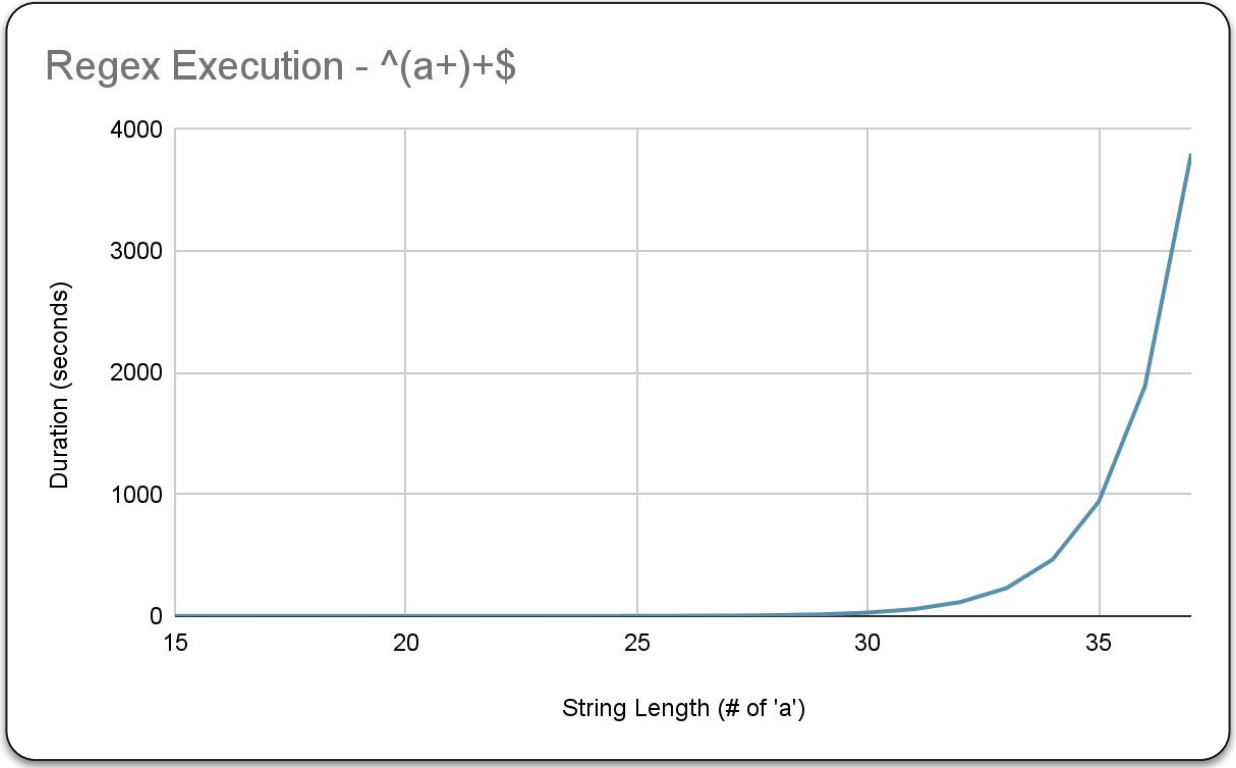
(a+)+ – The group itself is repeated one or more times

$ – End of the string

“a”	✅
“aa”	✅
“aaa”	✅
“ab”	❌
“ba”	❌
“”	❌

Malicious input: aaa…aaaX

Defenseless Programming - Input Validation - Regex



Defenseless Programming - Input Validation - Regex

Fixed Regex Pattern: ^a+$

- Removes the nesting

- Support a subset of regex for user supplied patterns

- Put a time bound around regex evaluations

Response Validation

Response Body

Server Error in '/' Application.

Maximum request length exceeded.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.Web.HttpException: Maximum request length exceeded.

Source Error:

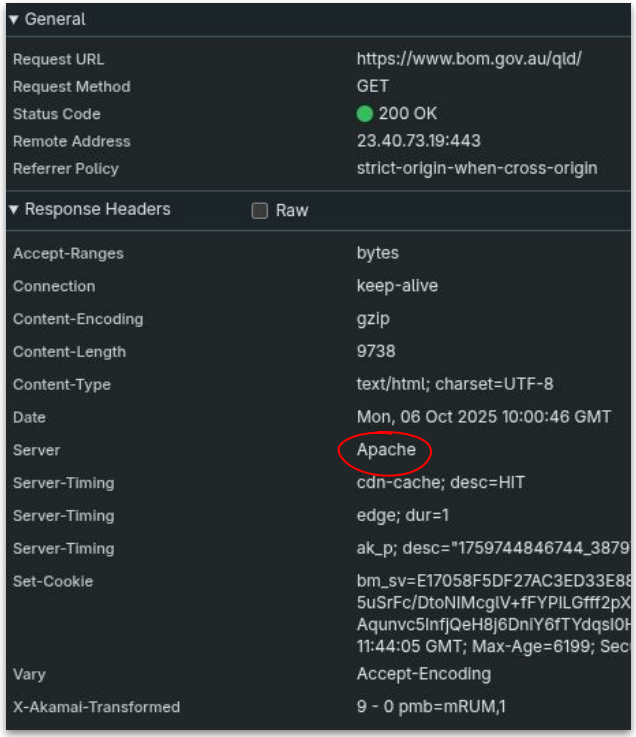
An unhandled exception was generated during the execution of the current web request. Information regarding the origin and location of the exception can be identified using the exception stack trace below.

Stack Trace:

```
[HttpException (0x80004005): Maximum request length exceeded.]
   System.Web.HttpRequest.GetEntireRawContent() +9899668
   System.Web.HttpRequest.GetMultipartContent() +63
   System.Web.HttpRequest.FillInFormCollection() +160
   System.Web.HttpRequest.EnsureForm() +69
   System.Web.HttpRequest.get_Form() +13
   System.Web.HttpRequestWrapper.get_Form() +14
   System.Web.Mvc.HttpRequestExtensions.GetHttpMethodOverride(HttpRequestBase request) +121
   System.Web.Mvc.AcceptVerbsAttribute.IsValidForRequest(ControllerContext controllerContext, MethodInfo methodInfo) +37
   System.Web.Mvc.HttpPostAttribute.IsValidForRequest(ControllerContext controllerContext, MethodInfo methodInfo) +39
```

Response Headers

- Modify
  - Server
  - X-Powered-By
  - …
- Include
  - Content-Security-Policy
  - CORS
  - …



Defenseless Programming - Lack of Authorization (IDOR)

GET: /orders/{**ORDER_ID**}

GET: /orders/4589d301-7f73-45e2-bbbe-e111a99b4fae

```
app.get('/api/orders/:orderID', async (req, res) => {
  const { orderID } = req.params;

  try {
    const order = await db.getOrder(orderID);

    if (!order) {
      return res.status(404).json({ message: 'Order not found' });
    }

    res.json(order);
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Internal server error' });
  }
});
```

```
app.get('/api/orders/:orderID', async (req, res) => {
  const { orderID } = req.params;

  try {
    const order = await db.getUserOrder(req.user.id, orderID);

    if (!order) {
      return res.status(404).json({ message: 'Order not found' });
    }

    res.json(order);
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Internal server error' });
  }
});
```

Secret Management

GitHub expands security tools after 39 million secrets leaked in 2024

By **Bill Toulas**

April 2, 2025 02:24 PM 0

GitHub announced updates to its Advanced Security platform after it detected over 39 million leaked secrets in repositories during 2024, including API keys and credentials, exposing users and organizations to serious security risks.

In a new report by GitHub, the development company says the 39 million secrets were found through its secret scanning service, a security feature that detects API keys, passwords, tokens, and other secrets in repositories.

"Secret leaks remain one of the most common—and preventable—causes of security incidents," reads GitHub's announcement.

"As we develop code faster than ever previously imaginable, we're leaking secrets faster than ever, too."

Secret Management - DO NOT HARDCODE!

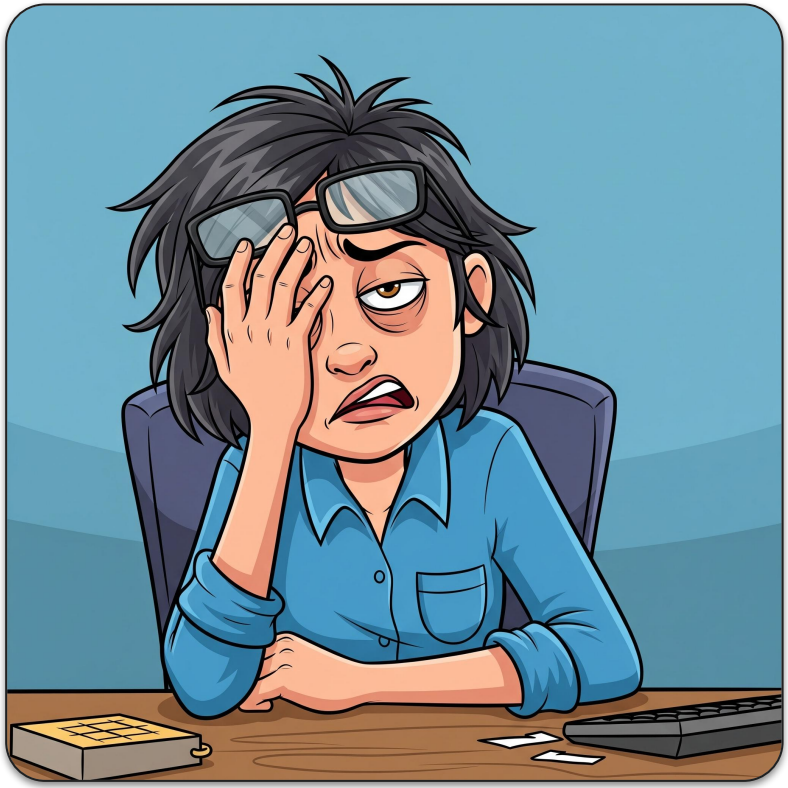
```
1   API_KEY = "c76258305d354beeb1c8ba676c12a4c3"
```

Secret Management - .env files

```
1  # .env.local
2  API_KEY=c76258305d354beeb1c8ba676c12a4c3
3  # .env.dev
4  API_KEY=c76258305d354beeb1c8ba676c12a4c3
5  # .env.staging
6  API_KEY=ce0fd20f75254c93a4e8856f68879338
7  # .env.production
8  API_KEY=72c96c2d25814391854cff64fb2c49ef
```

Secret Management - .env files

- They are still in your VCS…
- Add to .gitignore
- Still on filesystem…
- Still in history…

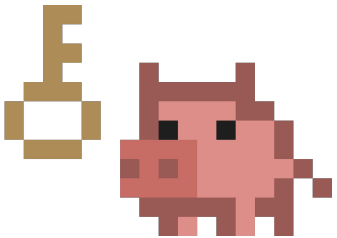


Secrets Management - .env

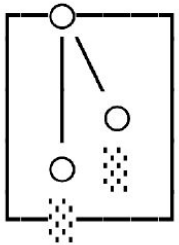
- Add logic to resolve the secrets in your application
- orchestrate startup
- Password managers
- Secret managers
  - Hashicorp Vault
  - AWS SSM Parameter store
  - GCP Secret Manager
  - etc

Secrets Management - VCS continuous evaluation

- Run secrets scanning as a pre-commit hook
  - Prevents secrets being committed and can be resolved quickly and easily
- Run secrets scanning on push (CI)
  - Secret has now been leaked but at least now it has been reported



Trufflehog



Gitleaks

Semgrep

Secrets Management - Git Cleanup

- Secrets have been pushed and now are leaked
  - Can’t just update the file, it’s in git history…

- Clean the git history
  - Get output from secret scanning tools
  - Use BFG to forcefully modify the git history

Secret Management - Rotate your secrets!

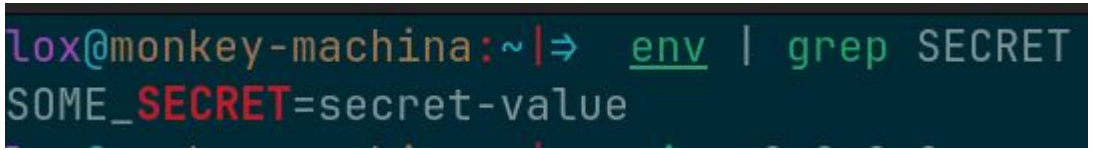
- Don’t wait for a leak
- Don’t wait for an employee to leave
- Bake rotation into your process from day 1
- Periodically rotate secrets

- Treat rotation as a insurance policy

Secrets Management - Shell profiles

- Retrieving secrets can be annoying
  - I'll export them in '.bashrc' so I never need to think about it!
  - Available for all my projects!

- Lots of stealers out there....
  - npm post install
  - pip
  - Dependencies
  - Malware



Secrets Management - Shell History

```
lox@monkey-machina:~|⇒  API_KEY=abc123 python3 app.py
```

```
lox@monkey-machina:~|⇒  grep "API_KEY" ~/.zsh_history
: 1748338636:0;API_KEY=abc123 python3 app.py
```

```
lox@monkey-machina:~|⇒   API_KEY=abc123 python3 app.py
```

Software Supply Chain

Software Supply Chain - Versioning

- Pinning to certain versions (v1.2.3)
  - If versions in the package manager are immutable

- Git commit (16ecab1875791e2b6ed0c9a6dae5a12a79d92120e1c3afbd3a9c8535ce44666d)
  - If package manager uses git tags

- Vendoring
  - Committing your dependencies
- Lock files

- Internal registries / mirrors

Software Supply Chain - Staying up to date

- Dependabot
- Renovate

- Follow CI/CD and automate patching

Software Supply Chain - Container base images

- It’s a container not a VM!
  - Do you need the bloat?
  - Do you need an OS?
  - Do you need a shell?
  - Multi stage build?

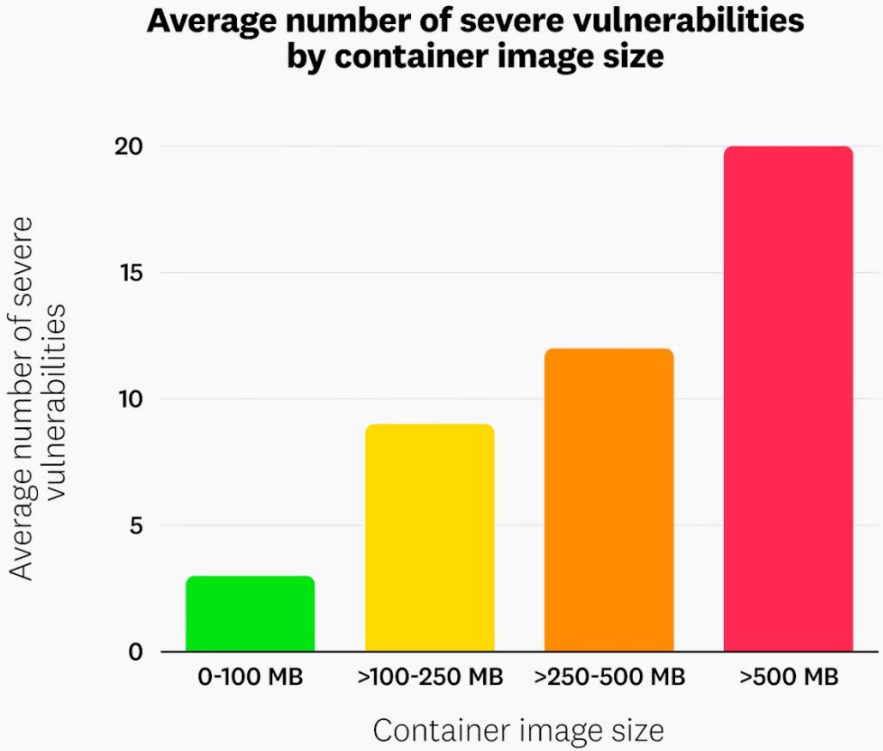
- scratch
- Alpine
- Google distroless
- Redhat Universal Base Image
- ${language}:slim
- Chainguard images

Software Supply Chain - Multi Stage Builds

Golang “Hello World!”	
Base Image	**Resulting Image Size**
golang:1.24.3-alpine3.20	308 MB
gcr.io/distroless/base-debian12	24.2 MB
scratch	2.21 MB
Raw binary	2.1 MB

USING SMALLER CONTAINER IMAGES LEADS TO FEWER VULNERABILITIES

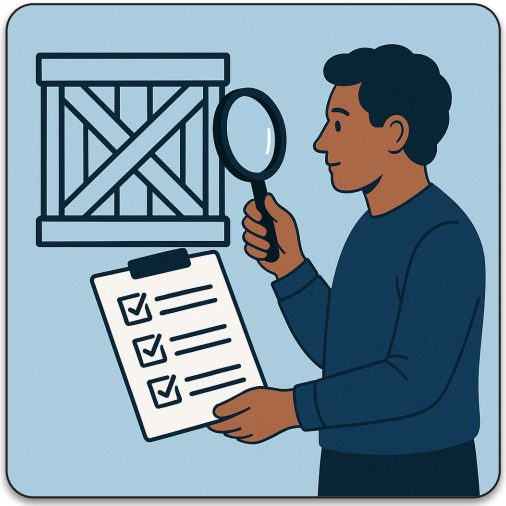
Source: Datadog



https://www.datadoghq.com/state-of-devsecops/

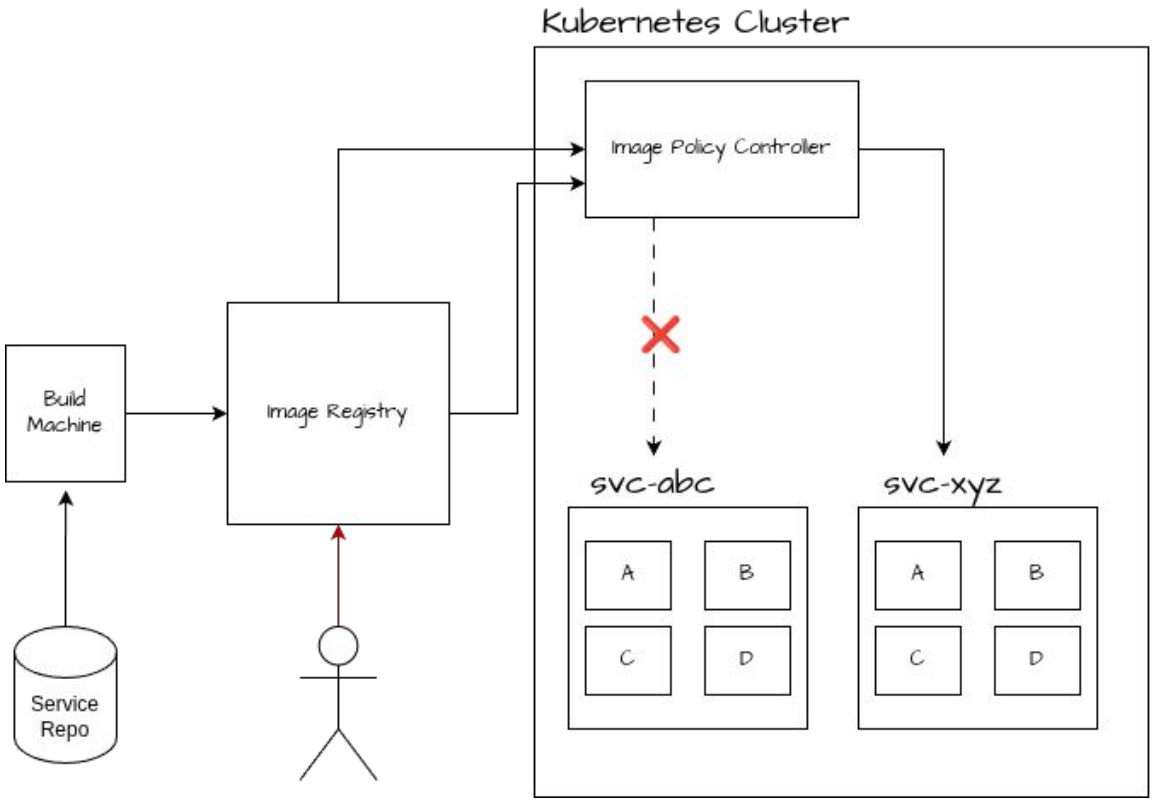
Software Supply Chain - Image Provenance and Attestation

- Sign images at build time
- Include metadata / labels
  - Compliance scans result
  - SAST
  - SBOM
  - Test coverage



- Reject the deployment if requirements are not met

Software Supply Chain - Image Provenance and Attestation




PII

PII - Don’t log it!

```
2025-05-25 16:00:40, 896 - __main__ - INFO - User did action:
User(id='266befc3-9d7d-49a4-acce-987df8514064',
 first_name='Olivia',
 middle_names=None,
 last_name='Kendrick',
 email='olivia.kendrick@example.com')
```

PII - Don’t log it!

```python
def mask(value: Any) -> str:
    if not value:
        return "None"

    raw = str(value)
    return raw[0] + "*" * 3
```

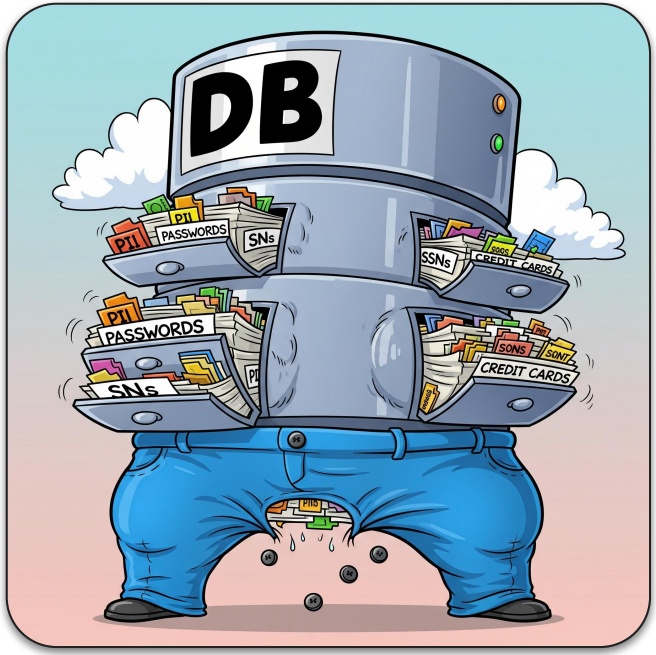
```python
@dataclass
class User:
    id: str
    first_name: str
    middle_names: Optional[str]
    last_name: str
    email: str

    def __str__(self):
        return (
            f"User[id: {self.id}, "
            f"first_name: {mask(self.first_name)}, "
            f"middle_names: {mask(self.middle_names)}, "
            f"last_name: {mask(self.last_name)}, "
            f"email: {mask(self.email)}]"
        )
```

PII - Don’t log it!

```
2025-05-25 16:02:57, 555 - __main__ - INFO - User did action:
User[id: f34a3603-22ac-445d-978b-e7a450ca9672,
 first_name: O***,
 middle_names: None,
 last_name: K***,
 email: o***]
```

PII - DB Access

- Do not share credentials
- Service accounts
- temporary user accounts
  - Scoped
  - Time bound

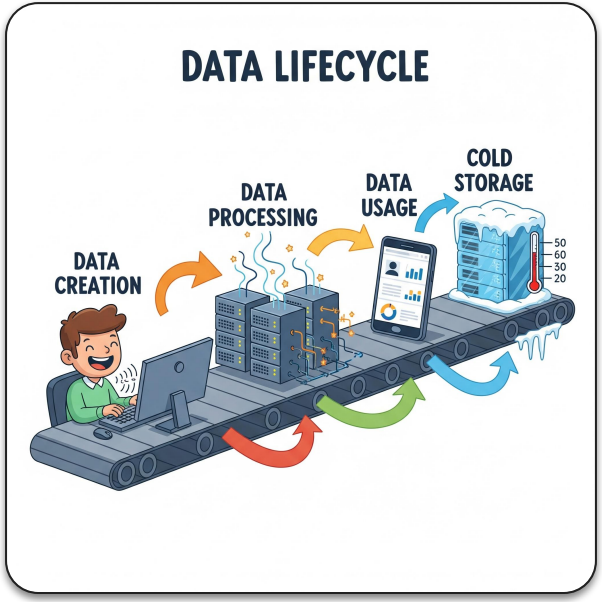


PII - DB Audit Logs

- Enable it!
- Security and Compliance
- Incident Response

PII - Data Lifecycle

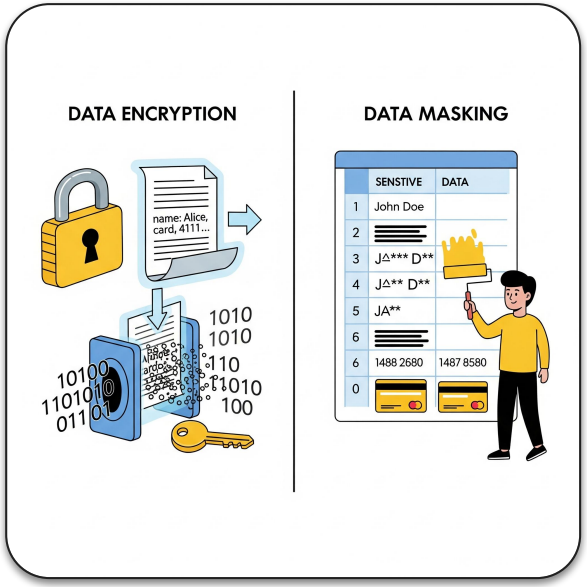
- Do you still need the data?
- Can you put the data somewhere else?



PII - Encryption or Data Masking

- Encrypt what you can
  - Hardware based if possible
  - Use a KMS from a cloud provider
  - Disk Encryption protects you from physical access

- Mask what you can
  - Names, emails, DoB, etc.
  - Just in time masking via proxies (identity aware)



Summary

- Spoke about a lot!

- Developing Software is hard
- Developing secure Software is harder

- Training is crucial
- Developers are incentivized to ship features

LachlanAshcroft.com/talk