Australia's National Science Agency

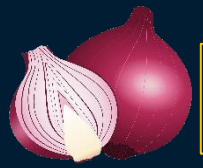
Cybersecurity for RSEs

Design patterns for eResearch web applications

Sharon Tickell | eResearch Australasia 2025

Disclaimer: the speaker is not a cybersecurity expert. Please fact check everything before you follow any recommendations made during this talk!

Networks have *Layers*...




Internet Standard (RFC 1122) aka TCP/IP

- Application Layer — HTTP, FTP, SMTP, Telnet
- Transport Layer — TCP, UDP
- Internet Layer — IPv4, IPv6
- Link Layer — Ethernet, VPN
- (Hardware)

OSI Model for Networked Communications

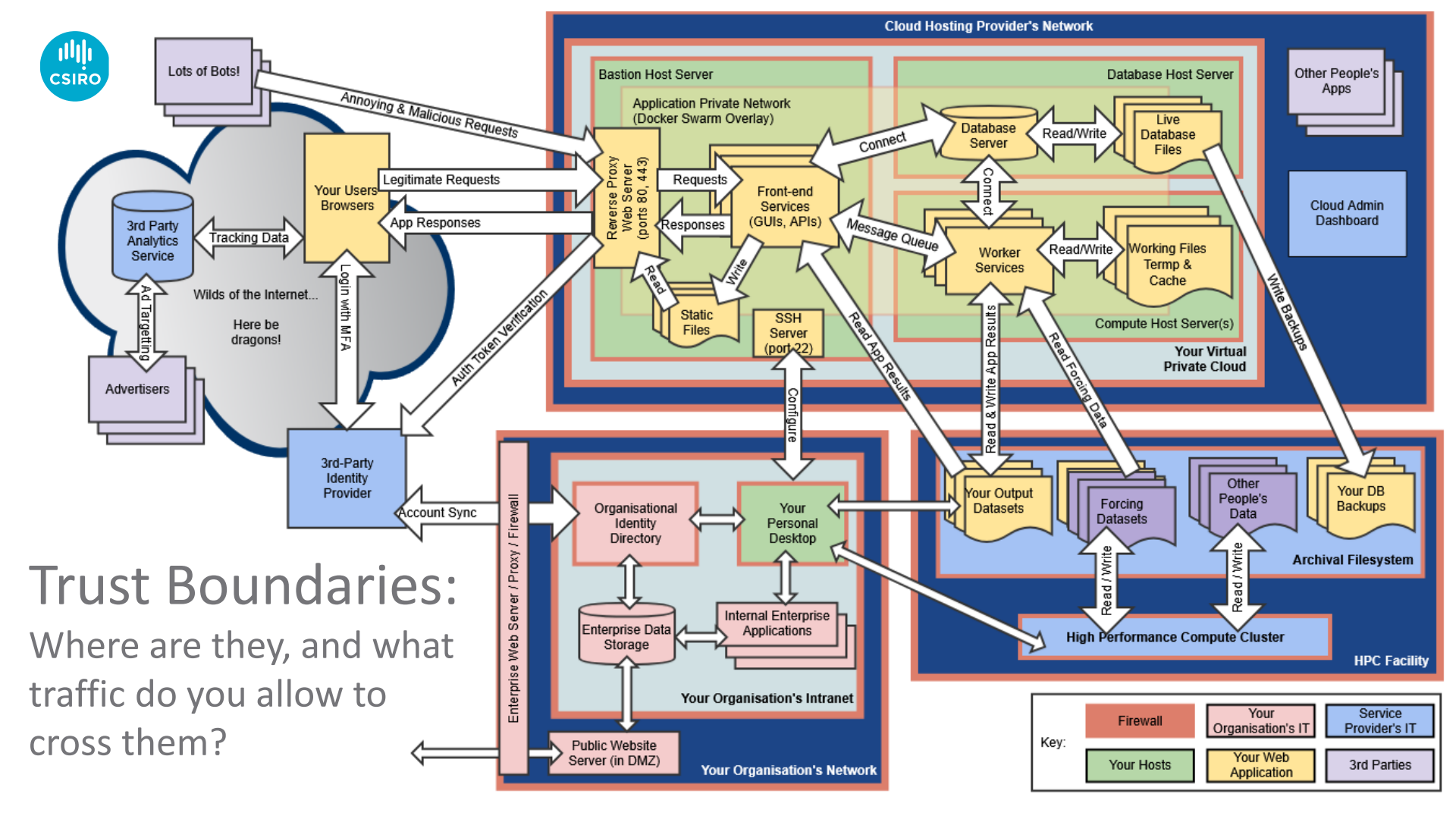
7	APPLICATION	Accesses protocols used by applications to communicate data to end users
6	PRESENTATION	Configures data into an acceptable format via translation, encryption, and compression
5	SESSION	Manages connections and the opening and closing of sessions between devices
4	TRANSPORT	Handles data transmission between devices using UDP and TCP protocols
3	NETWORK	Conducts routing between different networks, determining the most efficient pat
2	DATA LINK	Handles node-to-node data transfer between devices on the same network
1	PHYSICAL	Converts data into bits to carry it across physical network equipment

We are here!

User Accounts, Roles
(Non-standard design elements)

- OpenID, OAuth, SAML, LDAP
- TLS, SSL, Content-Types, Base-64
- Sockets, Cookies, CORS, CSRF
- Firewall: allow/block Protocol, Port
- IPsec, Firewall, DDOS-detection
- Ethernet, WiFi, Bluetooth identity
- Air-gapped Network

CSIRO



Trust Boundaries:

Where are they, and what traffic do you allow to cross them?

Strong Recommendation: Encrypt all the Things

Data in Transit:

- HTTPS for web traffic, always
  - TLS certs must be trusted
  - ACME certbot automated renewal
- Database connections which cross a trust boundary
  - OpenSSL KeyPair OK

Data at Rest:

- Sensitive information in your DB
- Sensitive information in data files
  - Includes database dumps!
- Session and Application cookies
  - Use a different key for every deployment and rotate often

DID YOU KNOW: Maximum TLS certificate lifespans are shrinking!

- (398 days until now)
- 200 days by March 2026
- 100 days by March 2027
- 47 days by March 2029

https://www.digicert.com/blog/tls-certificate-lifetimes-will-officially-reduce-to-47-days

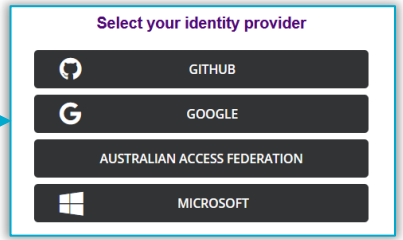
CSIRO

Authentication vs. Authorization

Authentication establishes *Identity by* testing for something your system knows about the connection:

- Local: Username + Password
  - This is 'traditional' local authentication
  - Password transmitted directly to app => risk
  - Passwords *stored* by app => more risk!
- Delegated: token issued by identity provider (IdP)
  - Caller authenticates with IdP (+ MFA!)
  - Caller includes token from IdP in requests to your application
  - App verifies token with IdP (did you really issue this?) and trusts the result
  - This workflow is OpenID Connect! (OIDC)

OIDC supports federation… and recursive federation



Authorization establishes *Privileges* – what is this identified user allowed to do?

- Likely to be very app-specific
- Role based authorization helps:
  - User ID -> Role Granted -> Privileges granted to Role
  - (Back end only needs to know about roles)

OAuth2 => App asks IdP for more account metadata about the user when verifying token. User must consent.

- e.g. email address, Full name, Institution name, ORCID, bio…
- Can be very useful for populating user profiles, deciding role grants…. But may also be personally-identifying information, so be careful

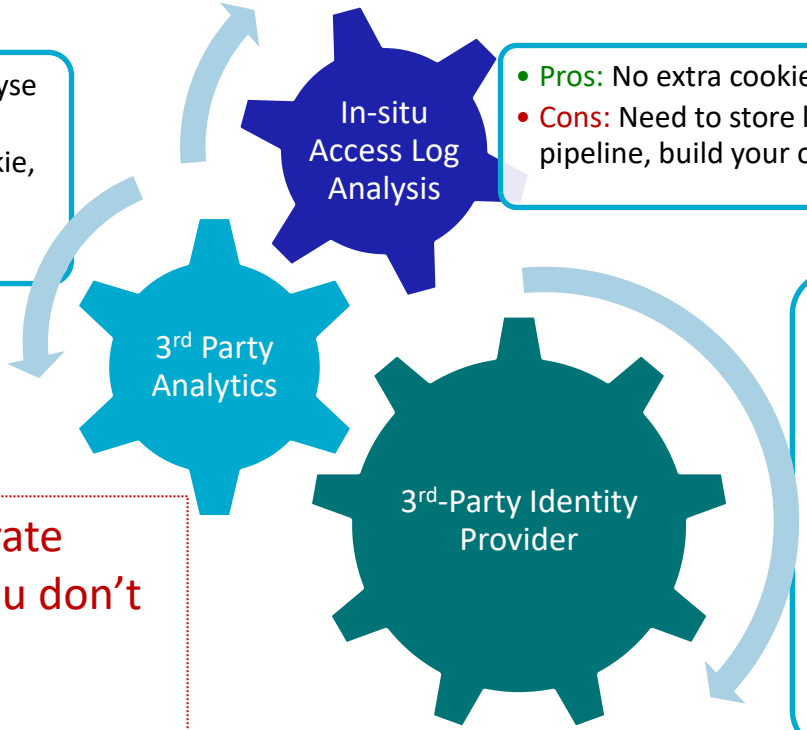
Privacy principles: store only what you really need

Design Pattern: auth transition at the boundary

Requests from Client	Boundary Transform	Upstream App Server
Arrive from client-IP To https://app-hostname/	• Terminate TLS (Decrypt Request Packet using SSL Private Key) • Set X-forwarded headers	From reverse-proxy-IP To http://upstream-host:app-port/ Plain text headers have original request info: • X-Forwarded-For (client's IP address) • X-Forwarded-Host (proxy-hostname) • X-Forwarded-Proto (protocol user requested)
Could include X-Token -* headers	No trust! Strip/ignore all X-Token headers included on client requests	-
Includes *encrypted* Authorization Header OR Cookie containing JWT with session info, identity claims	• Decrypt & Parse JWT. • Confirm ID, Role Assignment • Include *decrypted* info in well-known headers when request is relayed to upstream server	Can trust that these are legit ONLY if request arrives from known auth-proxy-IP on trusted network: • X-Token-Realm (which auth provider) • X-Token-Subject (unique ID => username) • X-Token-Email (way to contact this user) • X-Token-User-Roles (space-separated list)

The Security / Privacy / Simplicity Tradeoff

- Pros: Makes it easy to review & analyse traffic patterns
- Cons: Involves an extra tracking cookie, supports ad targeting, NOT essential under EU GDPR, people can and do block them



- Pros: No extra cookies needed for analytics
- Cons: Need to store log files, maintain analytics pipeline, build your own analytics reports

- Pros: No need to store password in your app database, Users don't need another password, MFA Support is SEP
- Cons: That authentication provider knows exactly what sites you visit and when.
- Neutral: Profile info is still personally-identifying information

The most secure and private information is the info you don't collect and store.

Design your application for a minimal footprint.

CORS and CSRF: Why don't my API calls work??

Cross-Origin Resource Sharing (CORS)

Restricts what a script *executing on a web-page in a browser* is allowed to request

- Preflight Request:
  - For unsafe requests (GET*, POST*, PATCH, PUT, DELETE)
  - Browser sends HEAD request
  - App issues OPTIONS response with CORS headers
  - Browser respects app restrictions in *real* request
- Response headers:
  - `Access-Control-Allow-Origin` (1 absolute URL or '*')
  - `Access-Control-Allow-Methods`
  - `Access-Control-Allow-Headers`
  - `Access-Control-Allow-Credentials` (not '*' origin)

Cross-Site Request Forgery (CSRF)

Is an exploit of the 'safe' methods (GET, POST) that CORS can't protect:

- Cross-origin Form-POST requests automatically get cookies for backwards-compatibility with HTML 4.0
- Technique 1: SameSite with set-cookie:
  - `SameSite=Strict` Skip cookie for all cross-origin requests.
  - `SameSite=Lax` Skip cookie for non-GET requests
- Technique 2: CSRF Tokens (many frameworks can help):
  - Issue token AND cookie on first GET request for page, store it in page context (so lost on refresh, page-back etc).
  - Token must be included with all subsequent requests, but NOT in URL or referred headers (those often leak in logs)
  - Token from header and decrypted cookie must match

Warning: Cross-site-scripting (XSS) can defeat all CORS and CSRF protection
MFA and human-in-the-loop is the most reliable safety check for now

NEW! Content Signals: https://contentsignals.org
Also https://github.com/creativecommons/cc-signals

The Barrage of Bots

- A website admin's job is not only to avoid malicious access attempts, but also nuisance ones
- In 2025:
  - Bots are now 20% of all social media accounts[1]
  - Bot requests are now 51% of all web traffic[2]
  - "Bad bots" are up to 37% of all web traffic[2]

Even non-malicious bots can overwhelm a small web application…especially if you expose data-download links

Keep an eye on your data egress costs!

To guide bots that behave: robots.txt

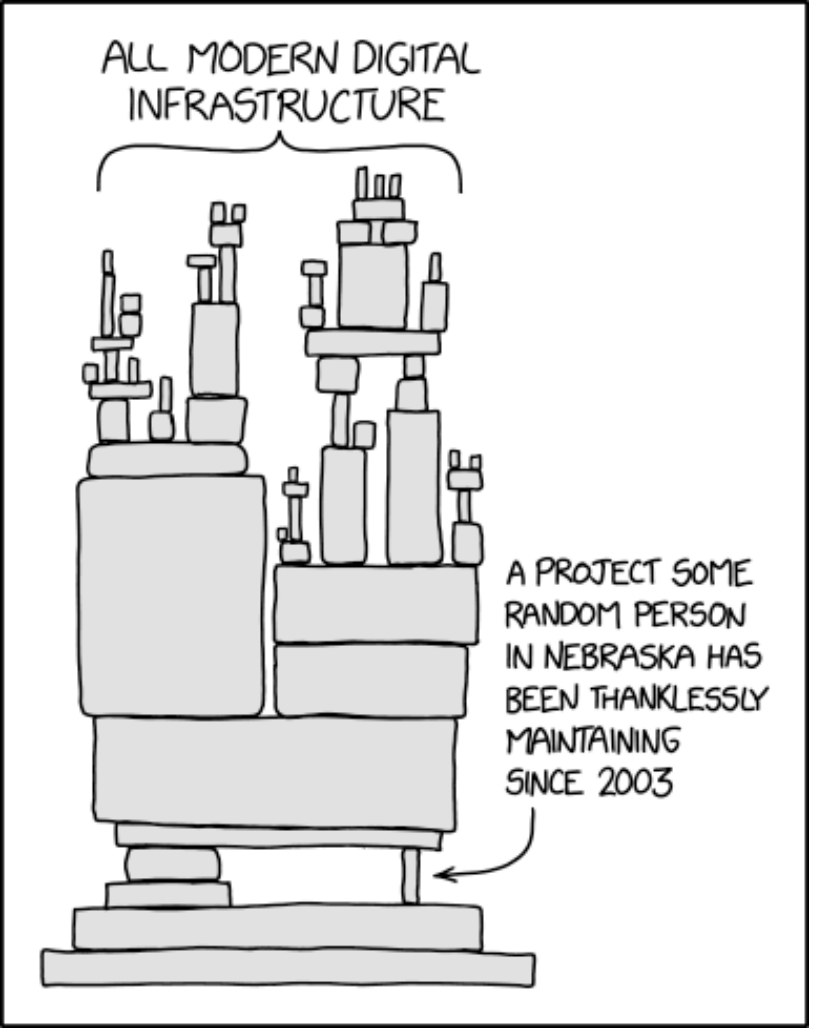
```
User-Agent: *
# Cloudflare model: Grants permission to index for
# Search and use with LLMs, but not AI Training
Content-Signal: ai-train=no, search=yes, ai-input=yes
# Creative Commons model: AI only if credit is given
Content-Usage: ai-use=n;exceptions=cc-cr
# Specify the path pattern as before
Allow: /

# Old-style blocks apply to both AI and search
User-Agent: *
Disallow: /thredds/
```

For bots that misbehave?

- Access log analysis + Fail2Ban
- automatically adds IP blocks to your firewall based on traffic patterns
- (Requires strict access log format)

1. https://www.nature.com/articles/s41598-025-96372-1
2. https://www.imperva.com/resources/resource-library/reports/2025-bad-bot-report/



Know your supply chain. **All** of your supply chain!

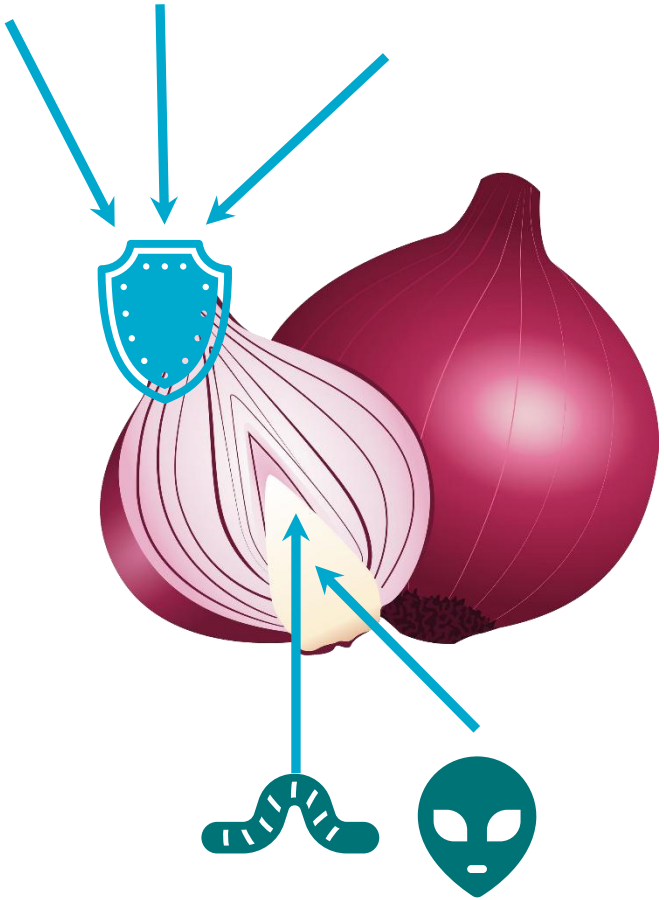
- Log4shell in 2021: had existed nearly 2 decades before detection/discovery
  - Disclosed Nov, Fixed Dec 6, CVE published Dec 9
  - Exploit scans started Dec 1, 60 variants targeted ~50% of all global corporate networks by Dec 14.
  - Still out there… many embedded systems could not be patched :/
- September 2025: Shai-Hulud Worm
  - NPM maintainer account compromised via phishing Sept 15
  - Malicious package scanned for secrets in repo history, added GH Actions workflows, made private repos public and propagated to more maintainer accounts…
  - Over 500 NPM packages impacted by Sept 16

What can you do? Detect-and-patch as fast as possible

- Automated scanning is your friend (don't ignore it!)
- Update everything often – both libs and OS

"Dependencies" https://xkcd.com/2347

Protect your Roots



The most dangerous compromise is the one that acts as you!

- Restricted runtime identity
  - (do not let your applications run as root)
- Be careful with the docker socket
  - (Beautiful flexibility, maximum risk)
- Only use minimum-privilege, minimum-lifespan API Keys
  - (and know how to rotate them fast)
- Never, ever commit a key to a git repo!

Failure is possible

- Can you detect it?
- Does your build automation help scan for supply chain issues?
- Do you know your reporting requirements?
- Do you know your risk profile?
- Do you have a safe backup?

Have an Action Plan:

- Shut it off
- Rotate Keys
- REPORT:
  - To your institution:
    - Often IT dept & integrity office
    - They can help!
  - To your cloud provider?
  - Office of the Australian Privacy Commissioner?
  - To your users…

Application security is also applied in layers:

Do you know all of yours?

Thankyou

Sharon Tickell
Senior Software Engineer
CSIRO Environment
Sharon.Tickell@csiro.au

Auth Protocol Disambiguation

Lightweight Directory Access Protocol (LDAP)

- 1990s-era protocol for accessing and managing directory services stored on a central server
- Has built-in authentication support => can be used to verify user-identity from back end via user-id / password combo
- LDAP auth is single-domain, requires on-prem LDAP server
- No token-based auth or cookie-based auth support

Security Assertion Markup Language (SAML)

- Single sign-on authentication protocol
- App redirects to SAML identity provider
- Identity Provider provides an XML-format *SAML assertion* to your app. (No authorisation support)
- Works cross-domain, Supports token-based workflows
- Can be implemented with LDAP as IdP back end

OpenID

- Federated *authentication*:
  - Support login workflow for multiple other applications
  - Authenticate users against multiple identity providers
- OpenID Server Examples: AAF (AU Research), Auth0 (paid), Okta (paid), KeyCloak (self-hosted)
- OpenID Client Examples: Keycloak, caddy-security

OAuth 2.0

- Controls and delegates *authorization* to access a protected resource
- provides API security through scoped access tokens (JWT => JSON)
  - Token Format
  - Identity Claims
- OAuth 2.0 server examples: GitHub, Google, Facebook… protected resource is your user profile

OpenID Connect (OIDC)

- Extends OAuth 2.0 with user authentication and Single Sign-On (SSO) to OpenID Identify Provider
- Lets you retrieve and store authentication information about your end users
- Defines several OAuth 2.0 scopes for user profile information.