University of BRISTOL

CyBOK

Cyber Security Body of Knowledge: Web and Mobile Security

Sergio Maffeis
Imperial College London

bristol.ac.uk

University of BRISTOL

CyBOK

The CyBOK project would like to understand how the CyBOK is being used and its uptake. The project would like organisations using, or intending to use, CyBOK for the purposes of education, training, course development, professional development etc. to contact it at contact@cybok.org to let the project know how they are using CyBOK.

bristol.ac.uk

Web & Mobile Security KA

CyBOK

- This webinar covers and complements selected topics from the "**Web & Mobile Security Knowledge Area - Issue 1.0**" document [WMS-KA for short]
- *"The purpose of this Knowledge Area is to provide an overview of security mechanisms, attacks and defences in modern web and mobile ecosystems."*
- We assume basic knowledge of the web and mobile platforms
  - The WMS-KA also covers some of the basic concepts assumed here

Scope

CyBOK

- The focus of WMS-KA is on the intersection of mobile and web security, as a result of recent *appification* and *webification* trends.
  - The KA does not cover specific mobile-only aspects including mobile networks, mobile malware, side channels.
- A full appreciation of web and mobile security requires familiarity with most others KAs from the CyBoK, and in particular:
  - Network Security
  - Software Security
  - Operating Systems & Virtualisation
  - Authentication, Authorisation & Accountability
  - Malware & Attack Technology
  - Cryptography
  - Human Factors
  - Physical Layer & Telecommunications
  - Hardware Security

Overview

CyBOK

- Security for the web and mobile ecosystem
- Authentication
- HTTPS and certificates
- Sandboxing
- Client side attacks
- Server side attacks

CyBOK

SECURITY FOR THE WEB AND MOBILE ECOSYSTEM

Appification

CyBOK

- *"There is an app for everything"*
  - On the mobile platform users interact with highly specialized apps dedicated to specific tasks
  - App functionality often depends on connecting to the web to retrieve/send data
  - Computation and complexity moves to the client
  - Many apps are entrusted with collecting, handling and communicating sensitive content
  - Large and varied attack surface, hard to secure
- Extensive tool and framework support lowers the barrier to entry
  - Even inexperienced developers with no software engineering knowledge can produce hugely popular apps
  - Negative consequences for the security of the ecosystem
- Appification of the desktop
  - Browser extensions are like apps for browsers
  - And so are desktop clients for popular web applications based on JavaScript
    - Electron and similar frameworks

Webification

CyBOK

- Significant increase in the proportion of applications built using web technologies
  - Python, PHP, Java, JavaScript (JS) on the server side
  - JavaScript on the client side
- Even mobile apps frequently rely on on the web platform
  - WebViews integrate web content in mobile apps
  - Apps can run JS and intercept web-related events
  - Web content can access data and sensors of the smartphone via JS APIs
- New attacks become possible
  - *App-to-web*: a malicious browser app can tamper with the web content being rendered in a WebView
  - *Web-to-app*: malicious web content accessed by a WebView can abuse the app API leveraging its permissions
  - Their composition: web content tricks an app to influence another app, and affect web content of the latter

App stores

CyBOK

- App stores are software distribution platforms that control what can be installed on what platform, for example
  - Google Play for Android
  - Apple's App Store for iOS and macOS
  - Chrome Web Store, addons.mozilla.org for browsers extensions
- App stores review and approve/ban apps based on reliability, compliance with store policies, and security vetting
  - Use of static and dynamic analysis techniques
  - Manual review when necessary
  - Apps are signed to identify developers and/or protect integrity
- Also users can review and provide feedback on apps
- Most users rely on these app stores
  - On some platforms it is possible to install apps and extensions directly, but it is relatively rare
- The app store trend has a significant positive impact on security
  - Drawing occasional censorship complaints

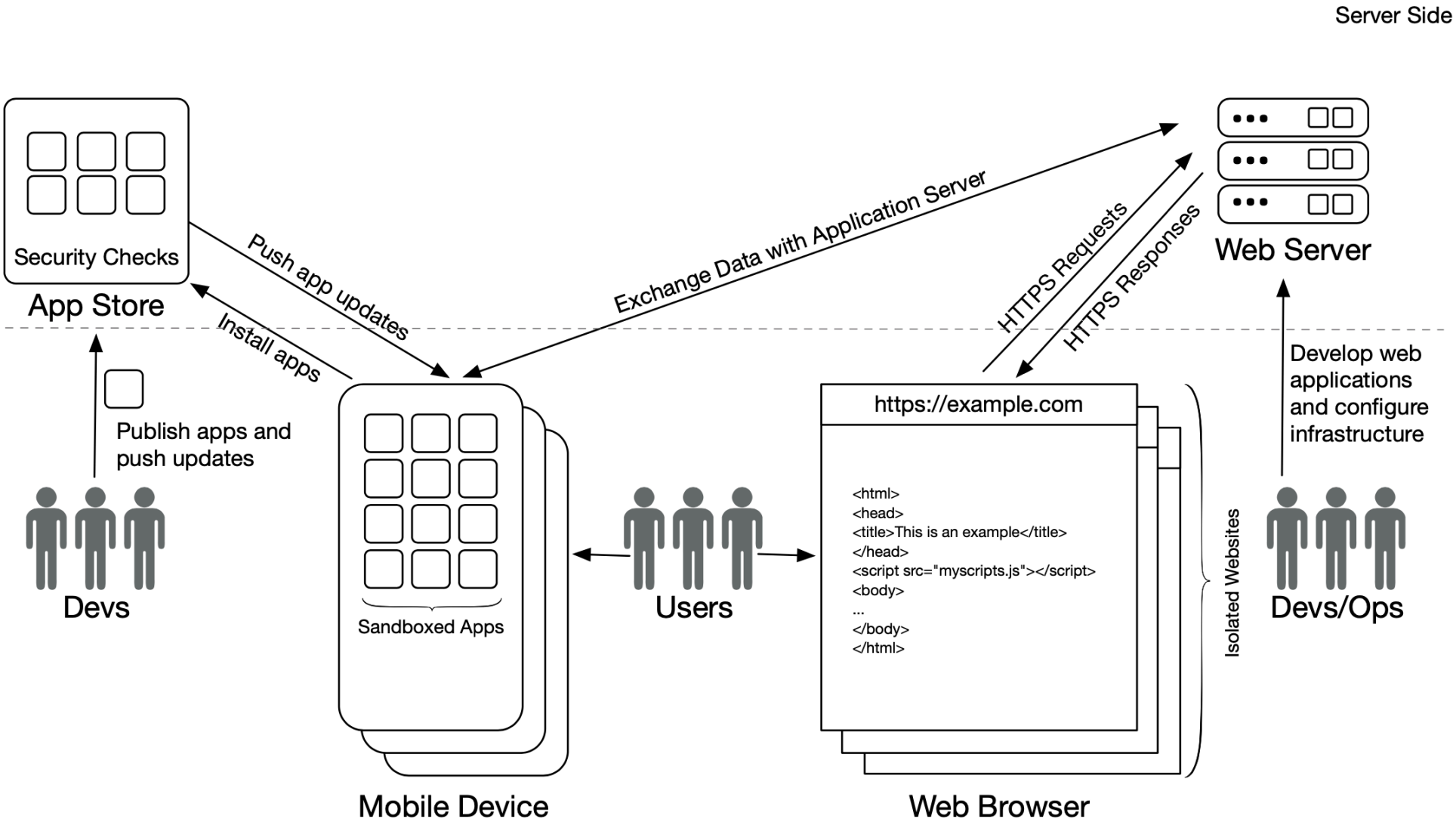
Security updates

CyBOK

- Software updates are a key mechanism to prevent attacks
  - Most security incidents follow the exploitation of known vulnerabilities, for which security patches have been released
  - Updating complex networks and systems can be time consuming and expensive
  - This provides a window of opportunity to the attacker
- The web/mobile platform occupies a sweet spot
  - Browsers and apps are mostly self-contained, and often connected to the web
  - Updates can be pushed directly to devices and security patches deployed much more quickly
  - Desktop browsers and most mobile apps update automatically (at least over WiFi)
    - App stores help coordinating updates
  - Operating systems have a longer release cycle
    - In particular “managed” (non-Google) Android releases often fall behind and fail to patch security issues in a timely manner.
- Third party libraries remain a challenge
  - Most apps and websites depend on libraries maintained by third party developers
  - Vulnerabilities may be inherited from libraries
  - App developers need to take extra care to update bundled libraries “disconnected” from their proper release cycle

Web and Mobile Ecosystem

CyBOK



[Diagram from WMS-KA]

AUTHENTICATION

Password based authentication

- Passwords are the prevalent mechanism to authenticate users on a device, or to a server
  - Some concepts discussed here apply also to other forms of authentication data (biometrics, etc)

1. Store credentials in a password file:

```
alice:wonderland
  bob:builder
charlie:brown
        …
```

2. User presents username and password
3. Check if username is present, and if so
4. Check that presented password is correct
5. Grant or deny access

- Password file is a very valuable target for hackers
  - Can impersonate any user in the system
- What if we encrypt passwords?
  - Decryption key must be available to support authentication, and can be stolen as well
- What if we hash the passwords?
  - *Offline dictionary attack*
    - Attacker builds a large dictionary of (hash,password) pairs
    - Looks for a stolen hash in the dictionary
    - If present, the corresponding password is *one* valid password for the target account

Salted hashes

CyBOK

Salted hashing:
- Salt: a cryptographically random string
- Picked at random, and looks random: not “0000000000000”
- Salted hash: Hash(plaintext|salt) = hashvalue

1. Store **salted hashes** of credentials in a password file
   - Format: username:**salt**:salted_hashed_password

```
alice:61C82:5C0E35473DA573EAE74B5A
  bob:8B4D8:C92A77164142EC14DC2F67
charlie:D9103:2D64320A38D8DE877AA1BD
...
```

4. Check that presented password is correct
   - Find user salt, see if Hash(password|salt) matches entry for that user

- Steps 2,3,5 as before
- Password file still a valuable target, but less so
  - Impractical to run a generic offline dictionary attack
    - A different dictionary is needed for each every possible salt
  - Offline dictionary attack against one specific user is still practical
    - Given salt for target user, build a targeted dictionary
    - But no benefit of sharing dictionaries among attackers

Online dictionary attack

CyBOK

- Users tend to use the same password on different websites
- Attacker submits username/password combinations to a running authentication system
- Usernames are easy to find or guess
  - May be email addresses, may appear in blogposts, may be people's surnames
- Previously used passwords are easy to find
  - Lists of passwords from hacked websites can be found in the public domain, or purchased on the dark web
  - Check if your password has been leaked: https://haveibeenpwned.com

Best-practices

CyBOK

- Use filters to ensure user selects long enough, random looking password
- Hash passwords using dedicated functions like `PBKDF2` or `bcrypt` to make it time-consuming to compute large dictionaries of hashes
- Don't ask user to change password often
  - Better to have a strong, long-lived password than tempt users to choose easy-to-remember ones
- After few failed login attempts for same username or from same IP
  - Slow down attempts: introduce artificial delay, display CAPTCHA
  - Ask an additional security question
- After many failed attempts, block user account, or requests from same IP
- Upon successful login
  - Show information about last login: user can report fraud
  - Notify user via email/sms if login is from unexpected machine/IP/location
- See also:

Enhancements

CyBOK

- Password managers
  - \+ Handle strong passwords for many different websites
  - \+ Help you avoid phishing sites
  - – Help you lose access to all of your accounts in one go
  - – Help hackers get all your passwords in one go
  - – Online password manager: exposed to hackers
  - – Offline password manager: potentially unavailable
- 2FA: *2nd factor authentication*
  - \+ Prevents attacks based on weak/stolen passwords
  - – You can get locked out of your account when 2nd factor is not available
  - – Has been shown to give false sense of security
    - Users choosing guessable pin because of reliance on 2FA
  - – Introduce new device/channel in your Trusted Computing Base
- OAuth and Single Sign On
  - \+ Authenticate user via trusted identity provider
    - Social networks
    - Enterprise *Single Sign On* providers: Okta, RSA Secure ID, Azure AD, ...
  - – Protocols and deployments not immune to hacks/flaws
    - High value identity may be compromised as a result of low-value sign-on

HTTPS AND CERTIFICATES

HTTPS

CyBOK

- HTTPS consists in running HTTP over an encrypted TLS connection
  - TLS provides confidentiality and integrity of the HTTP payload even against a man-in-the-middle network attacker
  - TLS prevents DNS spoofing attack
    - Attacker-controlled DNS server advertises malicious IP for target domain
    - Attacker is not able to create fake certificate for target domain to serve from that IP
- HTTPS (RFC 2818) is supported by vast majority of HTTP clients
  - Used by more than 90% of web traffic
  - Comparatively minor drawbacks
    - Some cost of using public-key crypto
    - Increased latency: first request to a website is slowed down
    - ISPs cannot cache HTTPS traffic
    - IDSs have limited visibility in traffic due to TLS
- Security issues
  - HTTPS runs in the browser, and a human controls the browser
    - Problems with accepting invalid certificates
    - Situation is improving as browsers are making it harder to bypass certificate warnings
  - Design and implementation bugs keep being found in TLS
    - Often with scary names: BEAST, CRIME, HeartBleed, DROWN, ROBOT, …

Certificate trust

CyBOK

- Certificates are normally signed by a known Certificate Authority (CA)
  - By default TLS clients trust a number of reputable CAs
  - Buying and renewing certificates from CAs can be expensive
  - https://letsencrypt.org initiative provides free certificates, and automated renewal scripts
- Self-signed certificates (SSC), two main options
  - Clients trust that a given self-signing key is appropriate to vouch for the domain covered by SSC
  - Clients trust a custom CA that signs the SSC
    - Used for to enable TLS traffic inspection by corporate IDS
- TLS protections against spoofing and MITM are ineffective when certificates cannot be trusted
  - Stolen private keys of certificates (Sony Hack, 2014)
  - Compromised CAs can sign spoofed certificates (DigiNotar 2011, Symantec 2015)

Mitigating rogue certificates

CyBOK

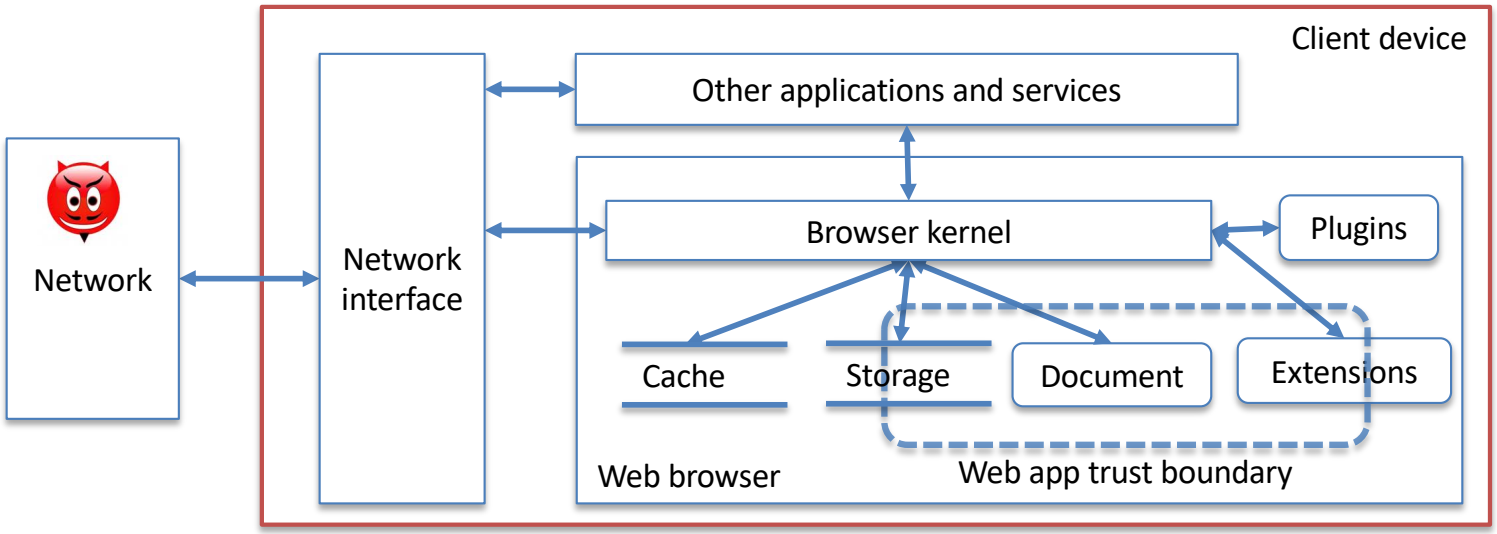
- Compromised and rogue CAs undermine trust in TLS
  - Can issue rogue certificates for legitimate domains
- Certificate transparency
  - All certificates created by complying CAs should be reported in tamper-resistant public logs
  - Domain owners can monitor logs, detect rogue certificates, and have them revoked
- DANE
  - Rely on DNSSEC to control trust in TLS certificates
  - Domain owner can deploy trusted self-signed certificates
  - Possible to restrict acceptable CA or certificate for a domain
  - Trust moves from CAs to DNS operators
  - TLSA DNS records
    - 0 – CA specification: trust this public CA
    - 1 – Specific TLS certificate: trust, but verify, this certificate
    - 2 – Trust anchor assertion: trust this new CA
    - 3 – Domain-issued certificate: trust this self-signed certificate

CyBOK

SANDBOXING

Sandboxing

CyBOK

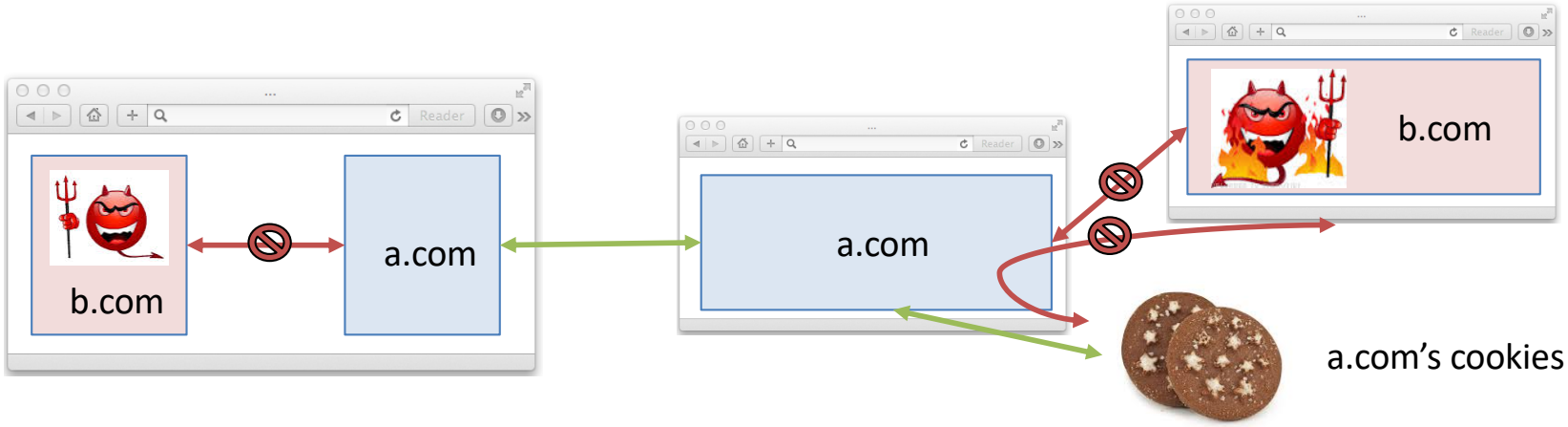


- Apps should be isolated from each other, and from the client OS
  - By default isolated processes cannot directly interact and share resources
  - Enforcement at the OS kernel level
  - Interaction between apps is mediated by controlled communication interfaces.
- Inside a web browser, web pages should be isolated from each other
  - Prevent low-level attacks
    - Leverage OS process-based isolation and sandboxing to limit effects of compromise
    - Each window/tab has its own process with renderer, JS engine, DOM
    - Browser kernel is the only privileged process with network access
  - Prevent attacks at the logical level via the Same Origin Policy (next slides)

The Same Origin Policy

CyBOK

- The Same Origin Policy (SOP) is the key security policy in the browser
- The URL https://www.example.com:54321/path?query denotes the origin (scheme=https,host=www.example.com,port=54321)
- Most resources are associated to the origin where they were loaded from
  - Scripts are associated to the origin of the page that loaded them
  - Browser storage (indexedDB, localStorage, cookies) are associated to origin of the page that was able to create them
- Main goals
  - Different pages can interact with each other if and only if they have the same origin
  - Storage can be accessed only by pages from the origin associated to it

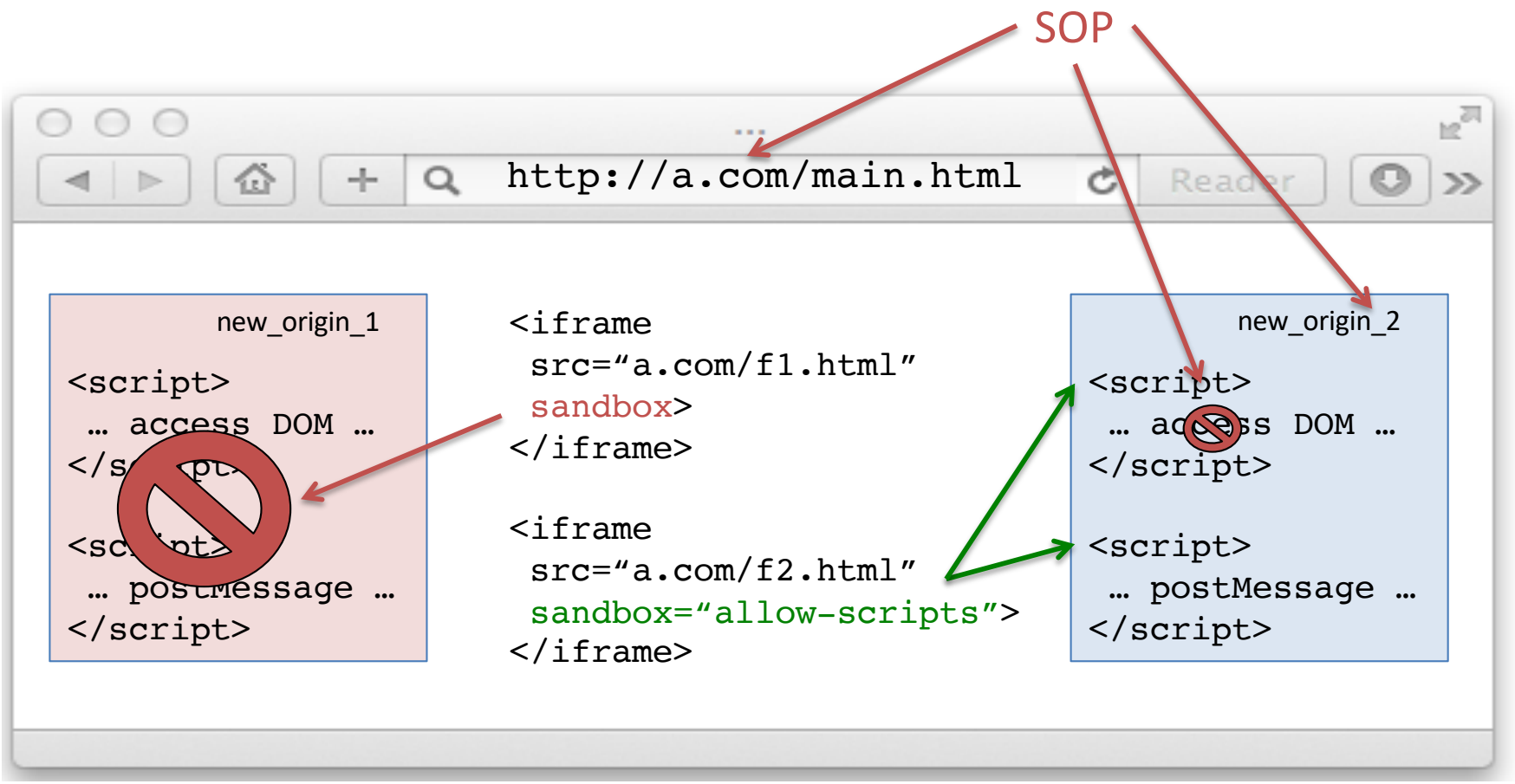


HTML5 sandbox

CyBOK

- The SOP can also be seen as too permissive for modern web applications
  - Same-origin iframe
    - May contain user-supplied content that is exposed to XSS attacks
    - Web app needs to restrict iframe access towards other more trusted iframes
  - Cross-origin iframe
    - May display advertising from a malicious provider with DoS attack on the whole page
    - Web app needs to restrict iframe ability to run JavaScript
- HTML5 `sandbox` attribute for iframes
  - Tells the browser to create a new unique origin and associate it to the iframe
  - All active behaviour is prevented by default in the sandboxed iframe
  - The SOP will prevent cross-origin access
- Relaxations
  - `allow-same-origin`
    - Does not segregate to new origin
  - `allow-{scripts/popups/forms/pointer-lock/top-navigation}`
    - Reintroduces the behaviour, as it would be allowed by the SOP alone

HTML5 sandbox

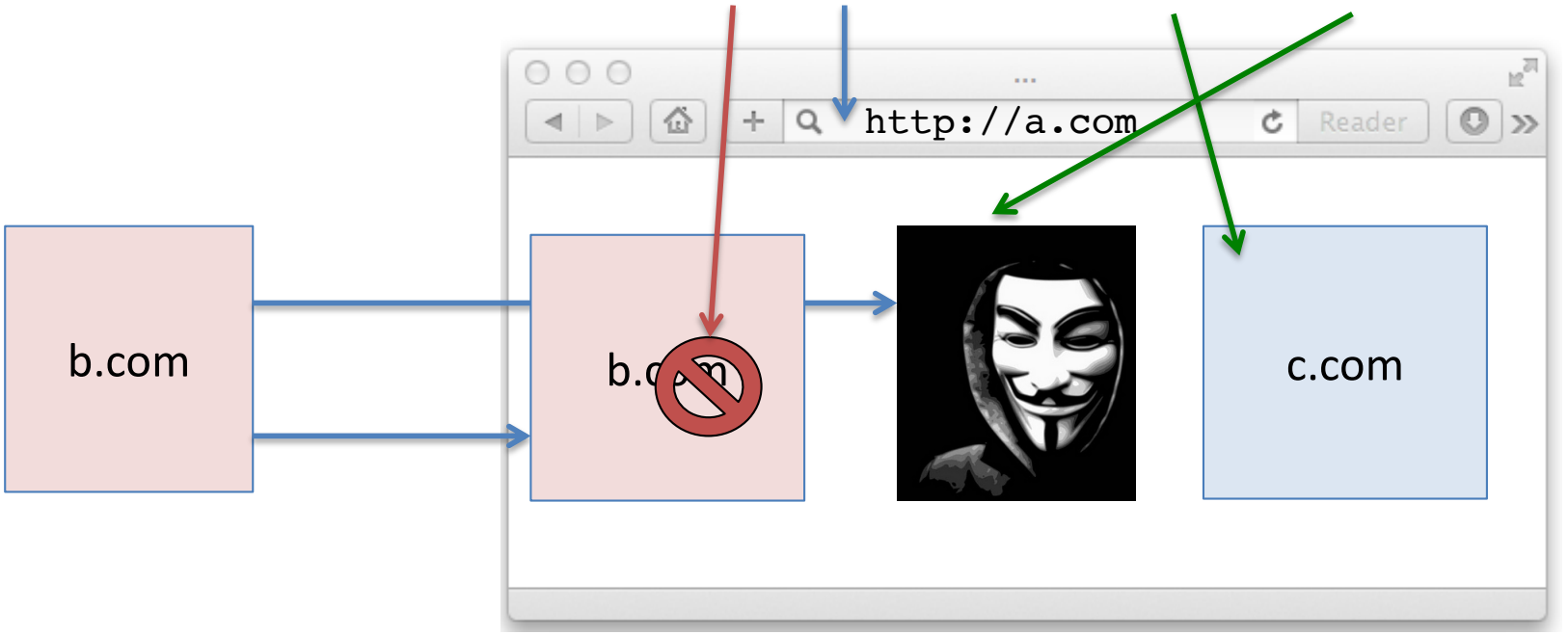


CSP

CyBOK

- Content Security Policy (CSP)
  - Server send a response header that tells browser a whitelist of what resources can be loaded and what scripts can be executed, and from where
  - Intended to mitigate mostly XSS, but also DoS
- Controls scripts, fonts, images, frames, media, objects, stylesheets, AJAX…
- Can be used to set `sandbox` attribute of loaded iframes

```
Content-Security-Policy: default-src 'self' http://c.com; img-src *
```



CLIENT SIDE ATTACKS

Cross-Site-Scripting (XSS)

CyBOK

- Probably the most common attack against web applications
  - An instance of injection attacks, related to SQL injection, command injection
- Conceptually simple
  - Attacker-controlled input makes its way to a trusted web page
  - There, it is executed as a script
- Critical
  - As if attacker controlled the whole origin in the browser
  - Other pages from the same origin are affected
  - Can access page resources: cookies, storage
  - Can send data to attacker
- Defenses
  - Validate inputs: accept only what you expect
  - Use anti-XSS filters
    - For example `htmlspecialchars()` in PHP
    - Be suspicious of overly-complicated regular expressions
    - Filters should be based on (audited) whitelists
    - Sanitization needs to be context-dependent: URL encoding, HTML entity encoding, SQL context, JavaScript/HTML context
  - Use templates or frameworks to validate inputs consistently
    - Similar to the use of prepared statements to prevent SQL injection attacks

DOM-based XSS

CyBOK

- A trusted script reads an attacker-controlled parameter and embeds it in the page
  - Injection vectors: URL, window.name, document.referrer, postMessage, form field
- Example
  - Intended usage:
    http://www.example.com/welcome.html?name=Daisy
    "Welcome user: Daisy"
  - Attack:
    http://www.example.com/welcome.html?name=<script>alert(document.cookie)</script>

```
<html>
<body>
Welcome user:
<script>
  var from = document.URL.indexOf("name=")+5;
  var to = document.URL.length;
  document.write(document.URL.substring(from,to));
</script>
</body>
</html>
```

Reflected XSS

CyBOK

- A an attacker-controlled URL parameter is embedded in the page by the server
  - In a regular response or in an error message
- Example
  - Intended usage:
    http://www.example.com/welcome.php?name=Daisy
    "Welcome user: Daisy"
  - Attack:
    http://www.example.com/welcome.php?name=<script>alert(document.cookie)</script>

```
<?
$name = $_GET["name"];
echo "<html>
      <body>
      Welcome user: $name
      </body>
      </html>";
?>
```

Stored XSS

CyBOK

- An attacker stores malicious data on a server, which later embeds it in user pages
  - Comments in a blog, user profile information, description of items for sale
- Example

store.php

```
<?
$conn = mysqli_connect("localhost","username","password","StoreDB");
$name = mysql_real_escape_string($_GET["name"]);
$desc = mysql_real_escape_string($_GET["desc"]);
$query = "INSERT INTO ItemsTable (name,desc) VALUES ($name,$desc)";
$result = mysqli_query($conn,$query);
?>
```

retrieve.php

```
<?
$conn = mysqli_connect("localhost","username","password","StoreDB");
$name = mysql_real_escape_string($_GET["name"]);
$query = "SELECT desc FROM ItemsTable WHERE name=$name";
$result = mysqli_query($conn,$query);
echo "<html>… Item description: $result …</html>";
?>
```

`http://www.example.com/store.php?name=MacBookPro` (inject payload on server)
`&desc=<script>alert(document.cookie)</script>`

`http://www.example.com/retrieve.php?name=MacBookPro` (deliver exploit to client)

Resident XSS

CyBOK

Target page:

```
<html><body>
Welcome user:
<script>
    document.write(localStorage.getItem("user_name"));
</script>
</body></html>
```

Attack vector:

```
localStorage.setItem("user_name","<script>alert('XSS!')</script>");
```

- Resident XSS (RXSS) is a variant of DOM-based XSS that exploits browser storage
  - Cookie, Web storage, Indexed DB, etc.
- Attacker must already be able to inject JavaScript to exploit the RXSS
  - **RXSS remains effective also after vulnerable page is patched**
    - Unlike DOM-based and Reflected XSS
  - RXSS cannot be detected by server, IDS, XSS Auditor
    - Unlike Reflected, Stored XSS
- Countermeasure: do not trust values stored in the browser (*defense-in-depth*)
  - Sanitise stored values like other user input
  - Periodically validate, refresh or delete stored data

Other XSS variants

CyBOK

- Self XSS
  - Users can be tricked into injecting malicious JavaScript in the page themselves , on the false promise that it's useful code
  - Browsers made it harder to paste JS in the address bar
  - Opening the browser developer console to run JS gives warning on prominent websites (try on Facebook!)
- Universal XSS
  - Victim visits (secure) target page and attacker page
  - Browser extension or browser's chrome have an XSS vulnerability
  - Attacker page exploits XSS in order to inject in target page
  - Example: CVE-2011-2107 in Adobe Flash Player
- Scriptless attacks
  - Victim switches off JavaScript to prevent XSS
  - Attacker still finds a way to inject CSS in target page
  - Using CSS, fonts, SVGs and plain HTML the attacker can read data from the page (credit card number, password) and exfiltrate it over HTTP

Phishing

CyBOK

- Typical steps
  - User visits a page controlled by the attacker that looks like a page form the target
    - Full replica of target HTML
    - Screenshot of target page, plus scripts that simulate interactive behaviour (like forms)
    - Different page that imitates target branding only
  - User reveals credentials or sensitive data to the attack page
  - Attack page forwards data to attacker and to legitimate page, or displays error message, or short acknowledgement
- Hosting phishing pages
  - Trade-off between control and legitimacy/reputation of the domain delivering the attack
  - Attacker domains
    - Name can have a similar spelling, possibly using unicode chars https://www.BankOfThe**VV**est.com (V+V , not W)
    - Or contain the target domain name: http://**paypal.com**.recovery-suspicious.com/
  - Compromise existing site to host phish: inherit the reputation of the compromised domain
    - Attacker may have limited control of the compromised domain
- Phishing countermeasures
  - Prevent spreading of links via spam
  - Train users to detect and avoid phishing
  - Safe-browsing (black)lists of major browsers include also phishing sites
    - See https://www.phishtank.com for latest reported phishing sites

Drive-by download

CyBOK

- User visits malicious page
- Page exploits vulnerability
  - Browser memory corruption (stack/heap overflow, …)
  - In plugins: Java, ActiveX, Flash, PDF…
  - In browser extensions (new!)
- Exploit installs malware on client machine
  - Or at least saves dangerous file that can be accidentally opened
- Countermeasures
  - Disable running plugins in the browser, or confine them to a restricted sandbox
  - Warn user visiting blacklisted websites (Google Safe Browsing, etc.)
  - Detect suspicious JavaScript with an IDS
  - Harden browser and OS against memory corruption vulnerabilities
    - Chrome-like architecture to isolate compromised processes
    - ASLR and other low-level defenses
    - Spectre/Meltdown exploitable via the browser using JavaScript

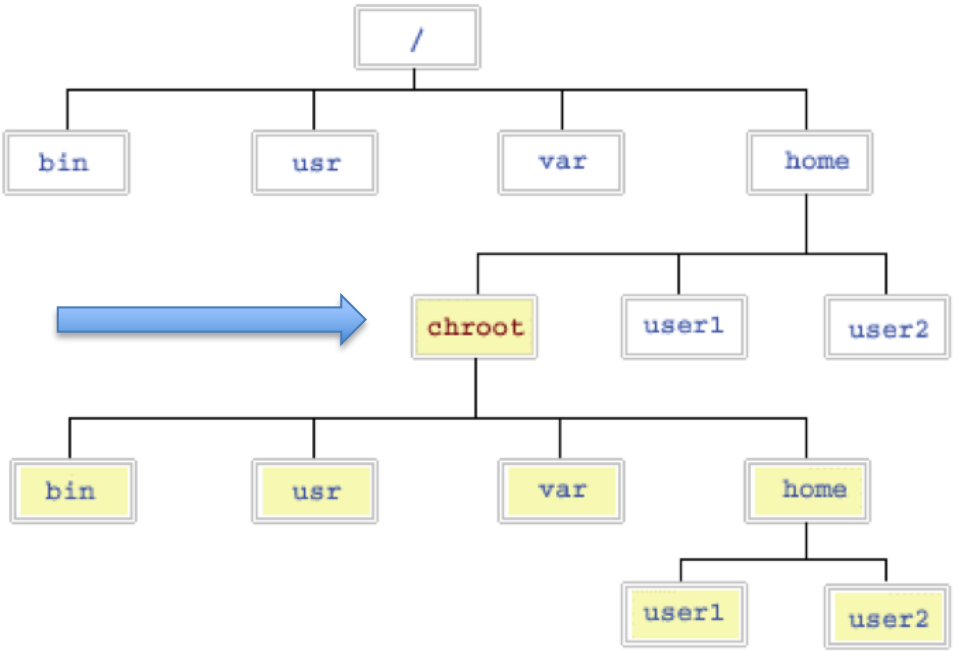
CyBOK

SERVER SIDE ATTACKS

Path traversal

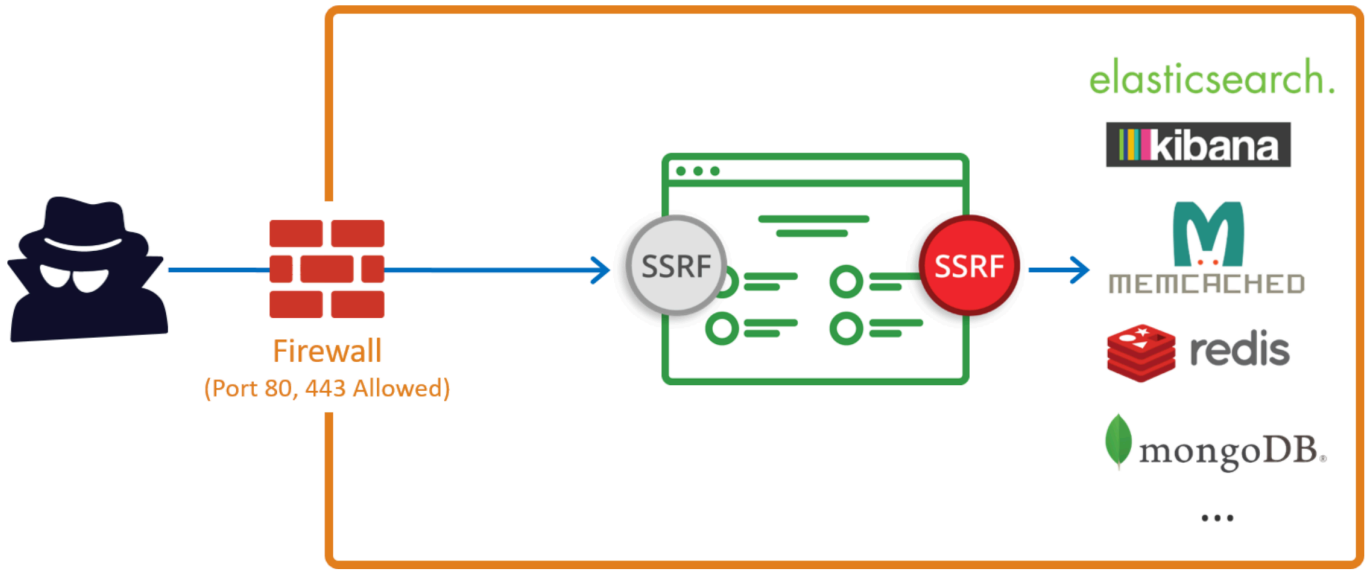
CyBOK

- Attacker input causes server to disclose unintended resource
- Examples
  - `http://www.example.com/../../etc/passwd`
  - `http://www.example.com/images/download.asp?name="../../etc/passwd"`
- General pattern
  - Server identifies resource based on user input
  - Attacker requests files likely to exist and unlikely to exist, and compares responses
- URL hacking
  - Attacker guesses path to a private resource
  - Crawling plugins available in most web app scanners
- Countermeasures
  - Special *www* user account for web app server with only access to public files
  - Web app process sandboxed to a virtual file system using "chroot jail"
  - Use access control restrictions in server configuration and/or web application logics



Server-side request forgery

CyBOK



(acutenix.com)

- Attacker controls parameter that becomes URL of request issued by the server
  - Server requests are issued beyond the firewall, have server privileges, may address the internal network
  - Examples:
    - Data exfiltration: `GET /?url=file:///etc/passwd HTTP/1.1`
    - Port scanning: `GET /?url=http://127.0.0.1:22 HTTP/1.1`
- Countermeasures
  - Should the user be able to provide URLs? Prevent poisoning of parameters (blacklist)
  - Whitelist requests that server-side application can issue
  - Don't handle unexpected responses

Command injection

CyBOK

- Command injection
  - Attacker input causes the execution of undesired commands on the server

```
http://example.com/ping_app/ping?ip=192.168.0.1;whoami
```

- Injection examples
  - Of PHP code:

```
$in = $_GET['param'];
eval('$out = ' . $in . ';');
```

  - Of shell commands:

```
$email = $_POST['email'];
$subject = $_POST['subject'];
system('mail  $email –s  $subject < /tmp/text')
```

- Countermeasures
  - Blacklisting: block inputs matching a list of forbidden patterns
    - Blacklists are *fragile*: attacker may find new dangerous parameters not in the list
  - Whitelisting: allow only inputs matching list of allowed patterns
    - Whitelists are more robust, but it is tricky to avoid false positives
  - Static and dynamic analysis, taint analysis in particular

Attacks on the application

CyBOK

- The application logics can be subverted
  - Design mistakes
    - User can bypass payment page and reach delivery page
    - Discount voucher can be used multiple times, or can be guessed
  - Authorization mistakes
    - Need for clear and expressive authorization policies
      - Hard to infer a posteriori during pentesting or code audit
    - Policy enforcement should be designed in the web application from the start
- Memory corruption
  - Attack the implementation language at the low level
  - Can lead to arbitrary code execution or denial of service attacks
  - Examples
    - Buffer overflows
    - Format-string abuse
    - Integer over/underflows
    - *Use-after-free*, *double-free*

Conclusions

CyBOK

- Takeaway points
  - Authentication is the first step into the systems, and must be done right
  - If HTTPS and certificates are used correctly we can focus our attention on the user, malicious web content and our server
  - Sandboxing our (web) apps is the key step to protect them from malicious apps and web content
  - Untrusted user input is the root of most evils
    - Validate, filter, sanitise!
- Refer to WMS-KA for more details, and additional topics not covered here
  - App permissions
  - Physical attacks on mobile
  - Other web attacks: SQL injection, CSRF
  - Server misconfigurations