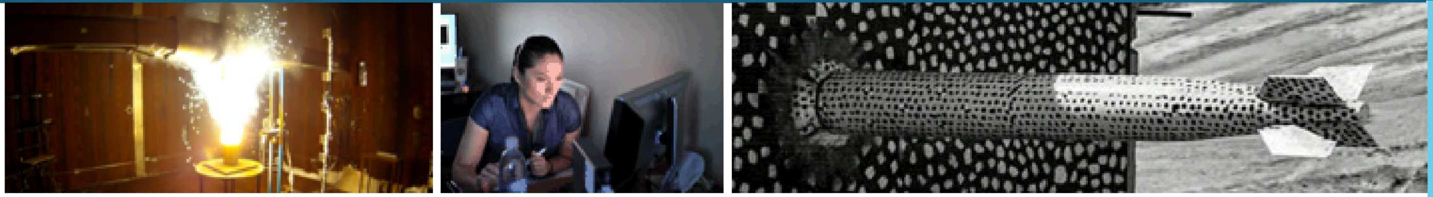
SAND2019-5132C

Cross-Site Request Forgery Challenges and Solutions



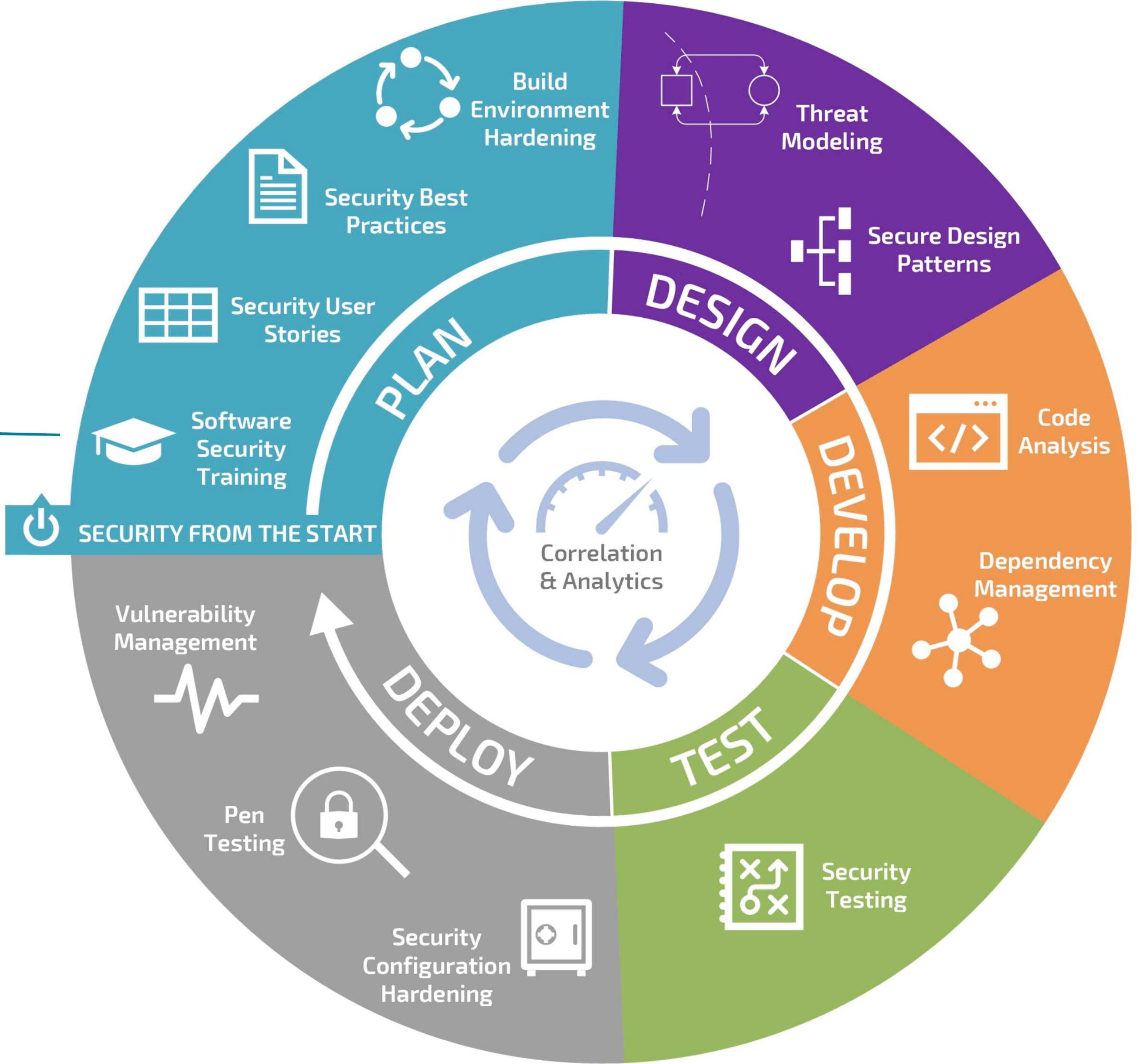
PRESENTED BY

Michael Coram, Sandia National Laboratories

mcoram@sandia.gov

Why Give This Talk?

Example of Security Awareness Training



Why This Topic?

Cross-Site Request Forgery (CSRF) is not one of the OWASP Top 10

- Listed as an "Additional Risk to Consider"

Genesis was a Project Lead asking our Secure Software Group about CSRF

- Developers identified that they lacked protections
- Project Lead wanted to understand whether they were needed prior to deployment
- If so, the Project Lead wanted implementation advice

This is EXACTLY what you want to see

What is CSRF?

CWE-352: Cross-Site Request Forgery Definition

"The web application does not, or can not, sufficiently verify whether a well-formed, valid, consistent request was intentionally provided by the user who submitted the request."

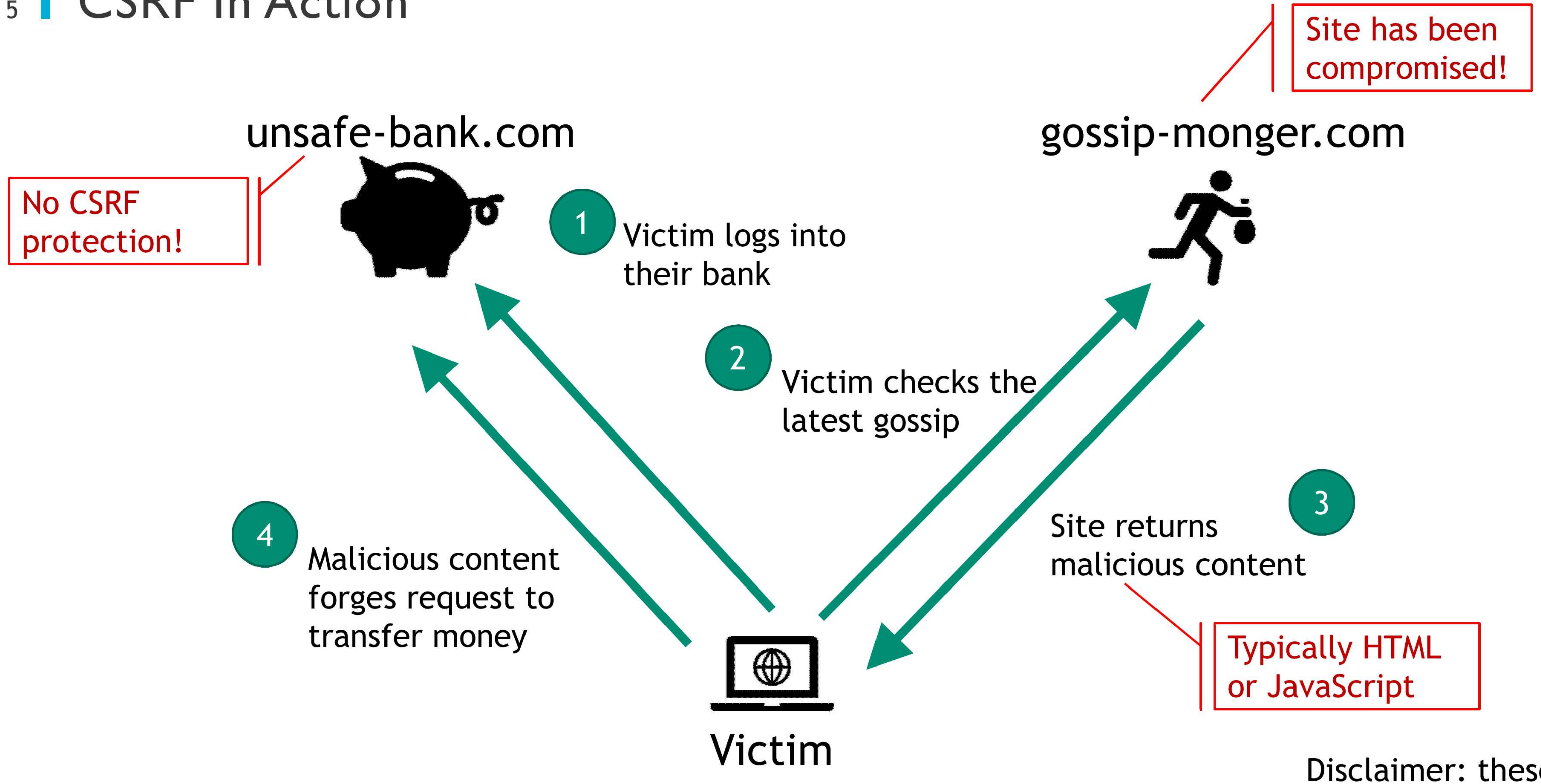
- https://cwe.mitre.org/data/definitions/352.html

OWASP Definition

"Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request."

- https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

CSRF In Action


Site has been compromised!
unsafe-bank.com
gossip-monger.com
No CSRF protection!
1
Victim logs into their bank
2
Victim checks the latest gossip
3
Site returns malicious content
4
Malicious content forges request to transfer money
Typically HTML or JavaScript
Victim

Disclaimer: these are not real sites.

Notable CSRF Attacks

ING Direct (2008)

- Allowed elicit money transfers[1]

Netflix (2006)

- Allowed an attacker to perform actions such as adding a DVD to the victim's rental queue, changing the shipping address on the account, or altering the victim's login credentials to fully compromise the account[1]

YouTube (2008)

- Allowed any attacker to perform nearly all actions of any user[1]

Paypal (2016)

- attacker [can] change a user's profile without permission[2]

2018 CVEs

- 461 CVEs mentioning CSRF, including Linksys Velop, boot2docker, and HP 2620 Series Network switches[3]

1. https://en.wikipedia.org/wiki/Cross-site_request_forgery, retrieved April 24, 2019
2. https://threatpost.com/paypal-fixes-csrf-vulnerability-in-paypal-me/119435/, retrieved April 24, 2019
3. https://www.cvedetails.com/vulnerability-list/year-2018/opcsrf-1/csrf.html, retrieved April 24, 2019

Example Attack Vectors

HTML Form submitted by tricking the victim into clicking a button or icon

- "Clickjacking" can be used to overlay the form submit button as a transparent image on top of a legitimate portion of the page

HTML Form that is auto-submitted via JavaScript

- <body onload="document.forms[0].submit" …>

JavaScript that uses AJAX to submit the data

- var x = new XMLHttpRequest();
- x.open("POST","https://unsafe-bank.com/transfer",true);
- x.setRequestHeader("Content-Type", "application/json");
- x.send(JSON.stringify({"account":"attacker", "amount":1000}));

In all cases, content from one site is causing a request to be sent to a different site

- The request is coming from the victim's browser, so appears to be legitimate

Same-Origin Policy

All modern browsers protect the user by preventing JavaScript from one site from reading content provided by another

- Malicious JavaScript from gossip-monger.com cannot access content from unsafe-bank.com
- Content includes the web page contents HTML and Cookies
- Does not include embeddable content such as images

This is why CSRF is considered a “blind attack”

- While the malicious content can send the request, it cannot read the response

Same-Origin Policy can be relaxed via Cross-Origin Resource Sharing (CORS)

- Allows cross-origin requests, including updates, via JavaScript
- By default, while JavaScript can make requests, CORS will prevent the requests from being fulfilled
- When properly configured, CORS will allow JavaScript requests from specific sites

Mitigating CSRF Attacks

The general approach is to send information that cannot be known by the attacker

- If unsafe-bank.com includes data in its requests that cannot be read by gossip-monger.com (due to the same-origin policy), then CSRF attacks will not success
- Typical approach is to provide a random piece of information (CSRF token) with requests that change state
- Requests that do not change state (e.g. GET) are not an issue for CSRF because the Same-Origin Policy prevents cross-origin reads

Assumes that this CSRF token cannot be guessed or otherwise known by the attacker

- Source of entropy matters if the token is randomly generated (which is typical)
- Communication channels must be secure to prevent eavesdropping (e.g. DNS spoofing)
- Other security vulnerabilities (especially cross-site scripting (XSS)) can be used to leak the CSRF token

CSRF mitigation options

CSRF Token per Request

- Each form sent to the browser has a unique token returned on submit

CSRF Token per Session

- Every update request to the server includes a unique token provided on login

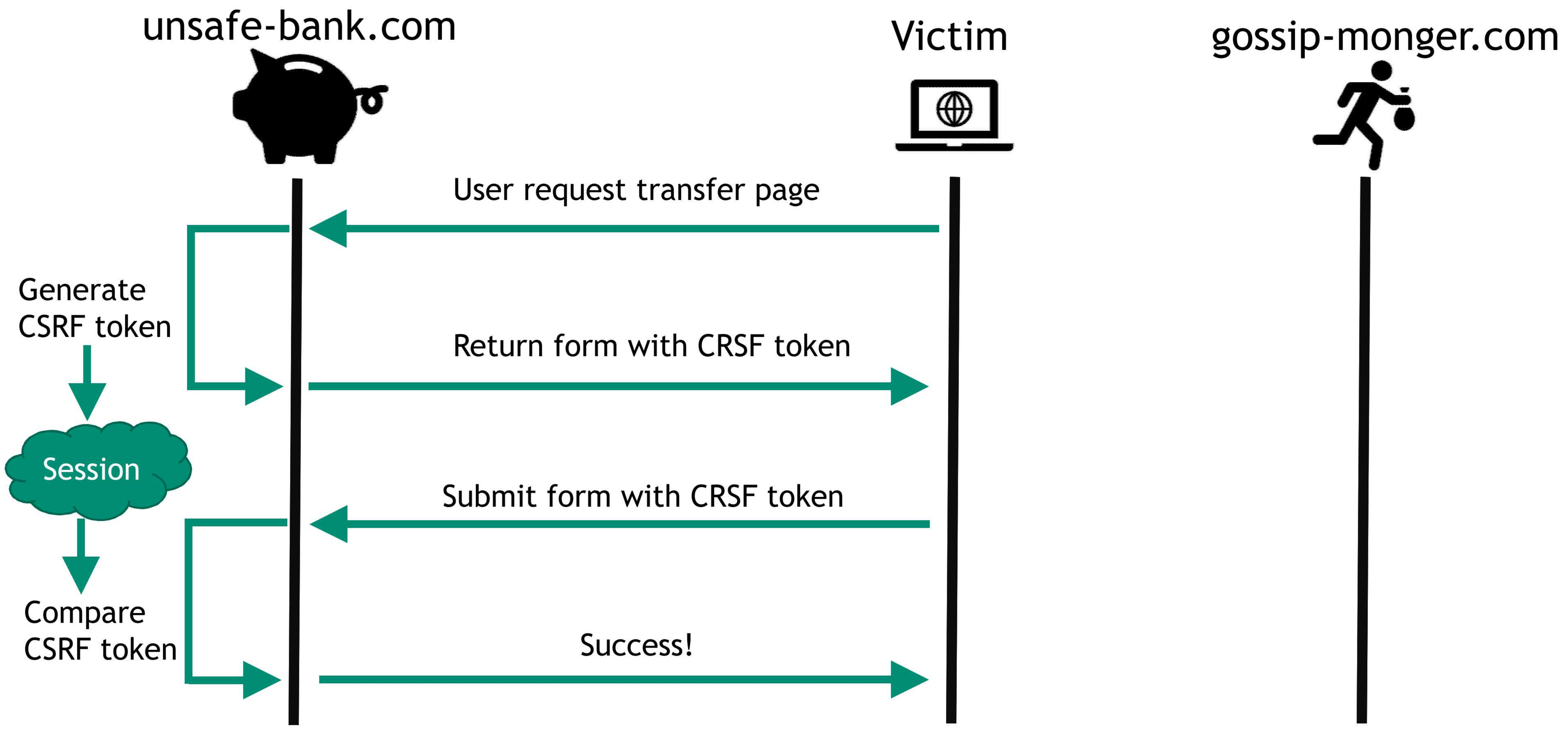
Double Submit Cookie Pattern

- Every update request has a CSRF HTTP Header compared to a CSRF Cookie

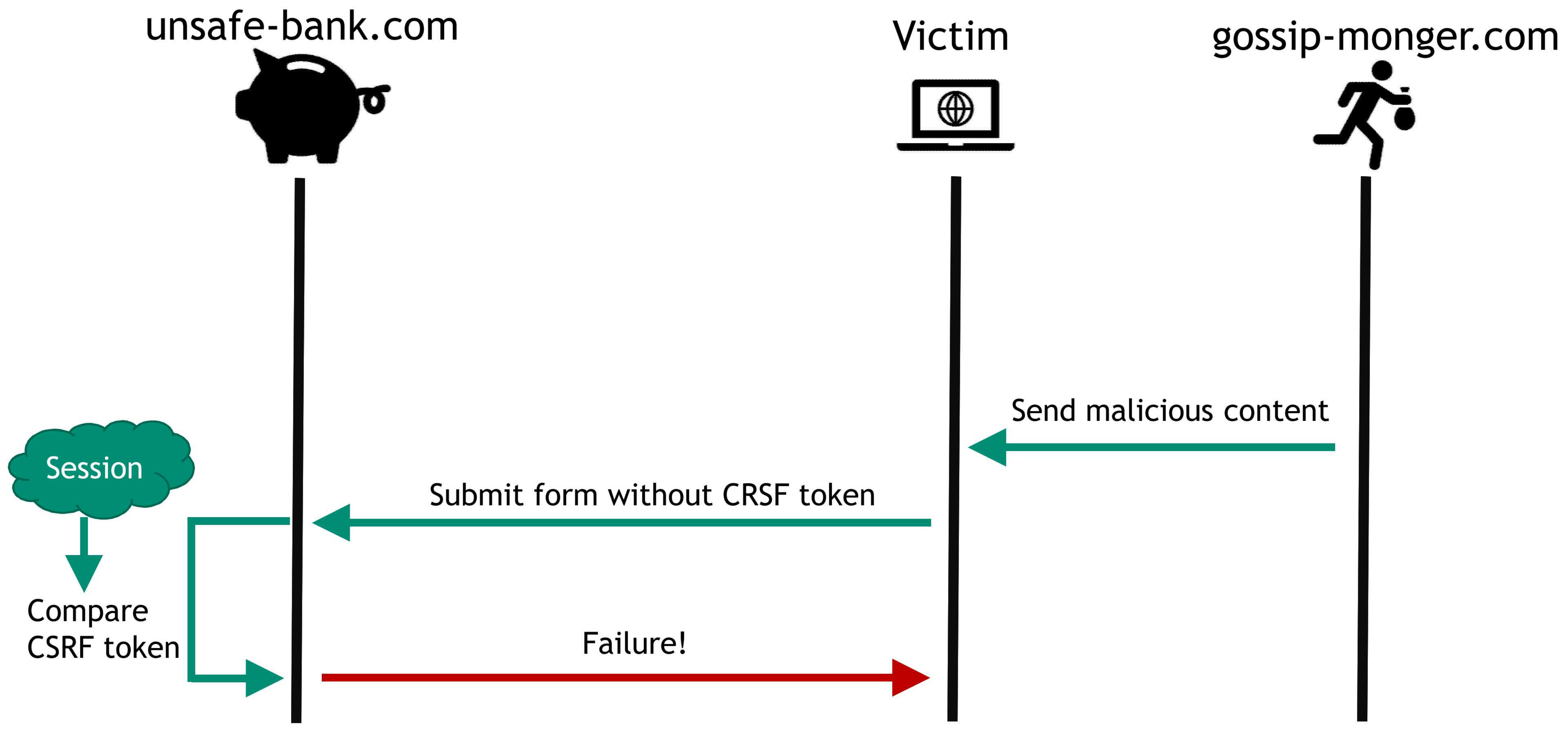
Same-Site Cookies

- Special configuration option for cookies that prevents cookies from being sent with requests generated by other sites

CSRF Mitigation: Token per Request



CSRF Mitigation: Token per Request



CSRF Mitigation: Token per Request

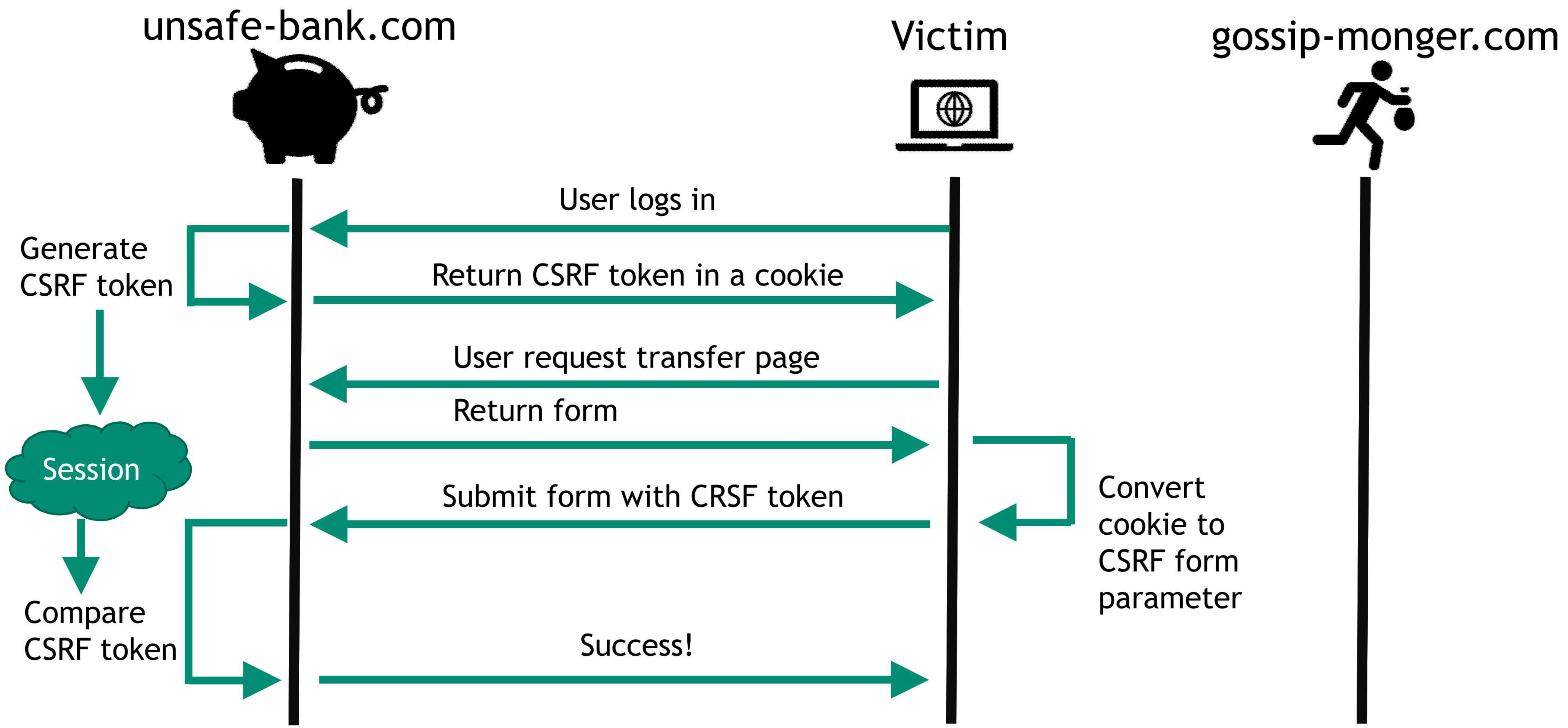
Pros

- Most secure option as unique token is created for every form
- Prevents the same form from being resubmitted, preventing replay attacks
- Supported by many web frameworks

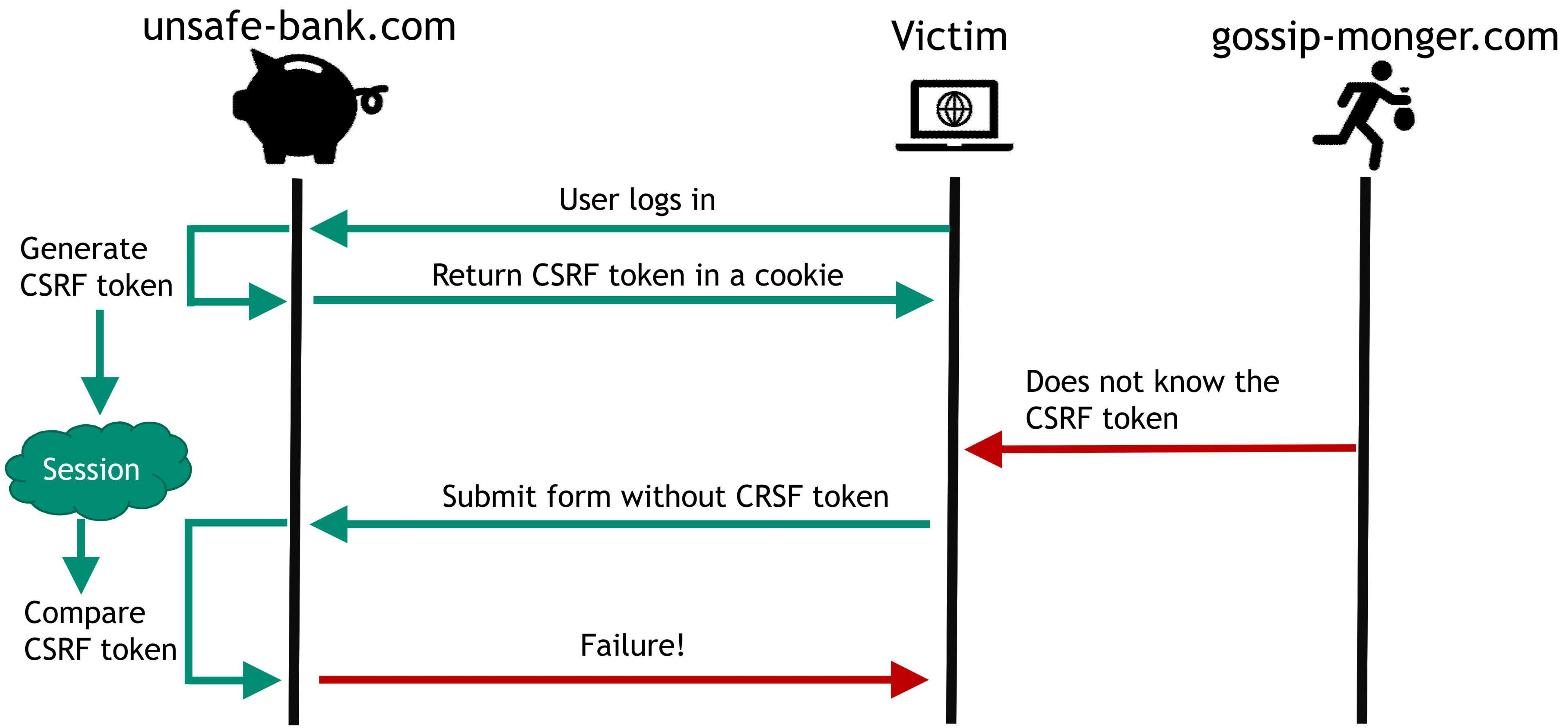
Cons

- Back button will not work (which in some cases may be desirable)
- Requires request / response pattern, which is not typically how Single Page Applications work
- Requires that the server keep state (each form and its associated CSRF token)

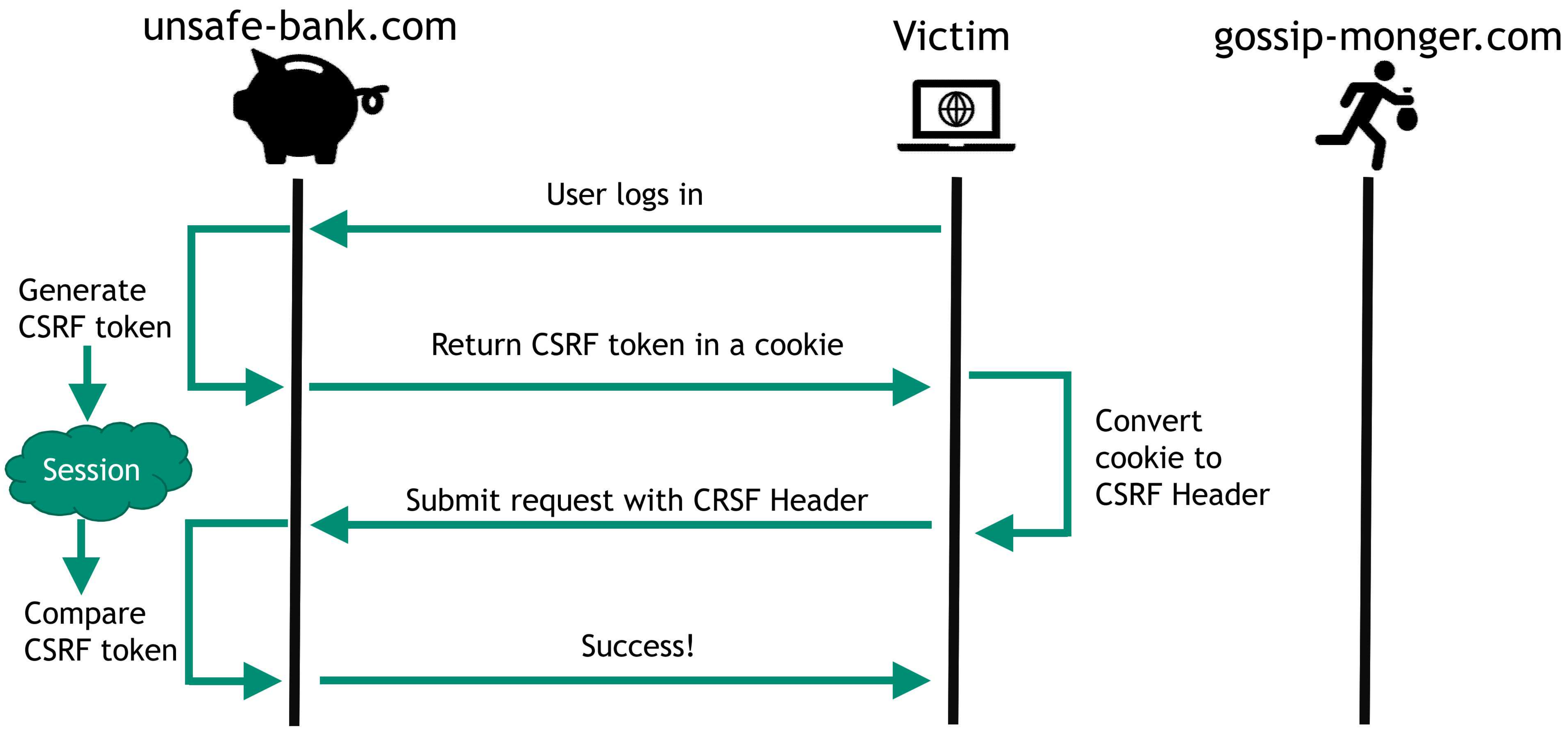
CSRF Mitigation: Token per Session – Form Based


unsafe-bank.com
Victim
gossip-monger.com
User logs in
Generate CSRF token
Return CSRF token in a cookie
User request transfer page
Return form
Session
Convert cookie to CSRF form parameter
Submit form with CRSF token
Compare CSRF token
Success!

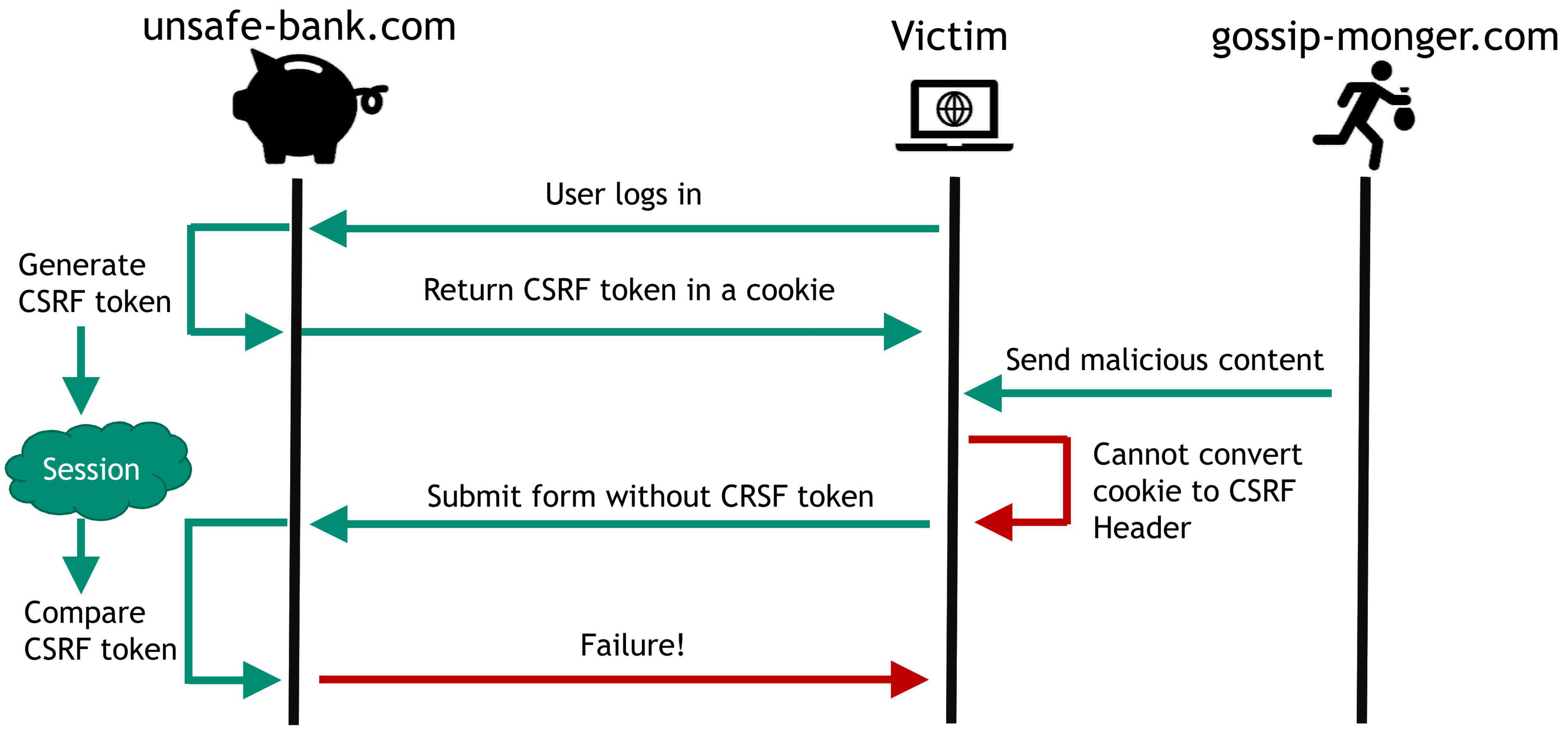
CSRF Mitigation: Token per Session – Form Based



CSRF Mitigation: Token per Session – JavaScript Based



CSRF Mitigation: Token per Session – JavaScript Based



CSRF Mitigation: Token per Session

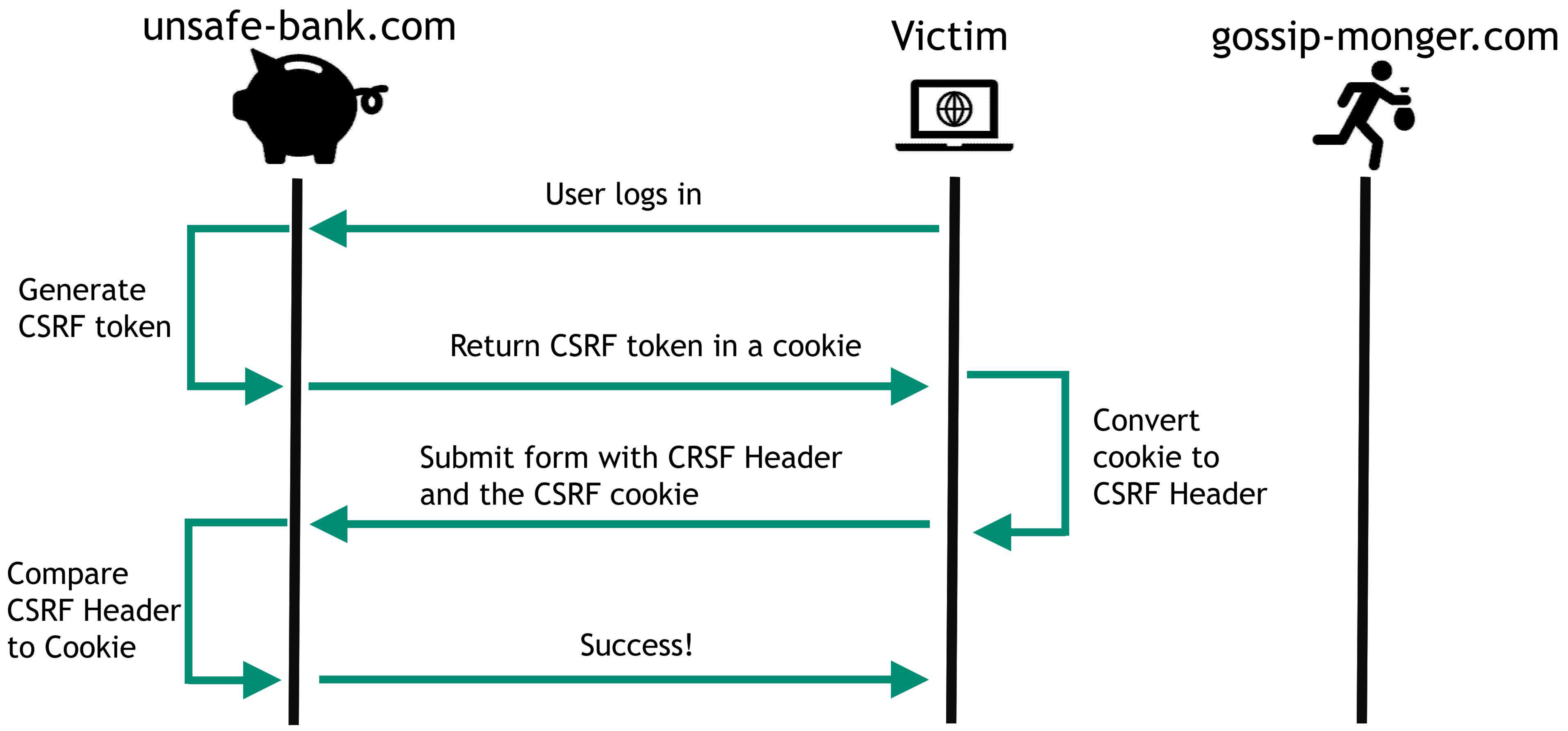
Pros

- Works with Single Page Applications
- Back button works for form-based applications
- Supported by many web frameworks

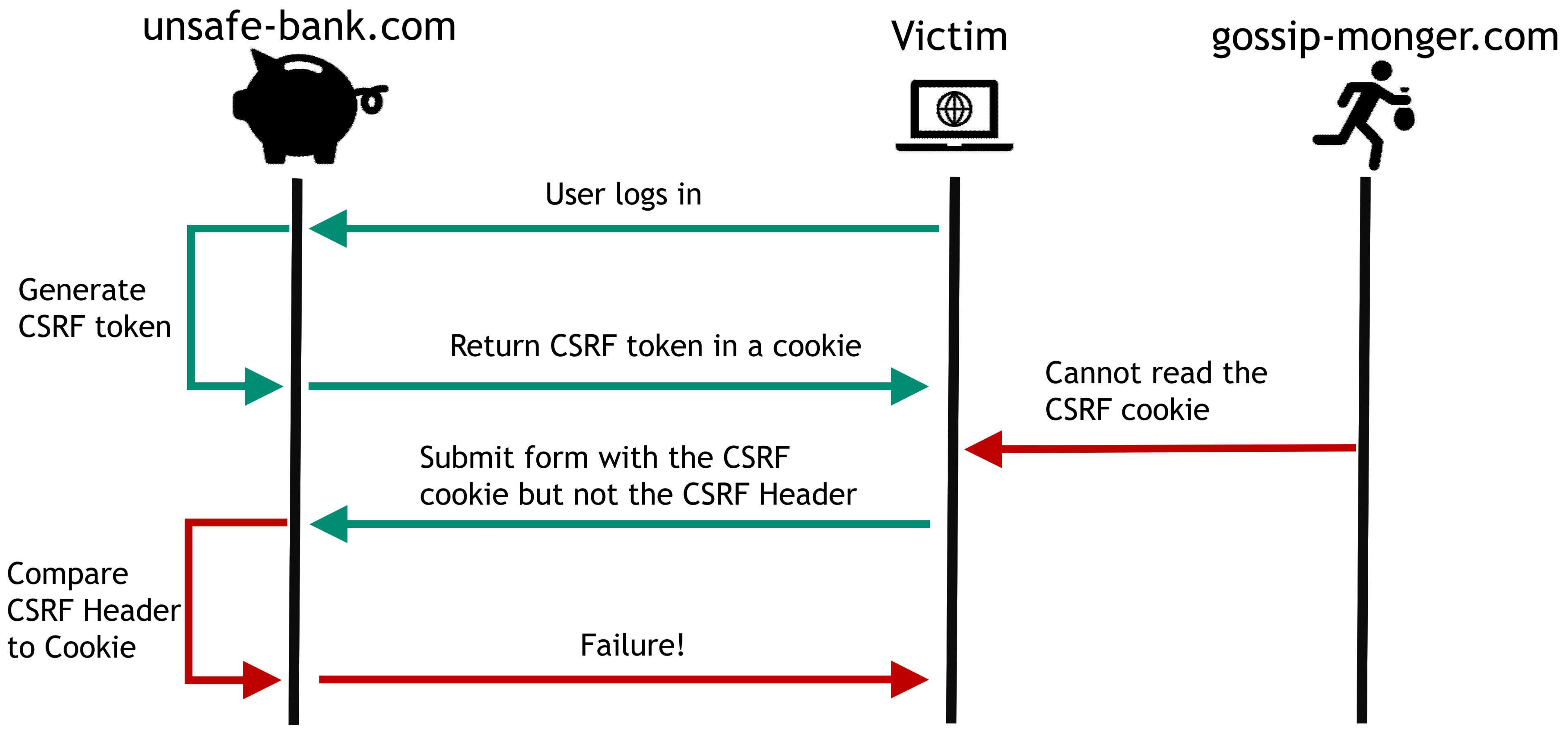
Cons

- Lifetime of the CSRF token is longer, providing more time for an attacker to subvert
- Requires that the server keep state (each session and its associated CSRF token)

CSRF Mitigation: Double Submit Cookie


unsafe-bank.com
Victim
gossip-monger.com
User logs in
Generate CSRF token
Return CSRF token in a cookie
Convert cookie to CSRF Header
Submit form with CRSF Header and the CSRF cookie
Compare CSRF Header to Cookie
Success!

CSRF Mitigation: Double Submit Cookie


unsafe-bank.com
Victim
gossip-monger.com
User logs in
Generate CSRF token
Return CSRF token in a cookie
Cannot read the CSRF cookie
Submit form with the CSRF cookie but not the CSRF Header
Compare CSRF Header to Cookie
Failure!

CSRF Mitigation: Double Submit Cookie

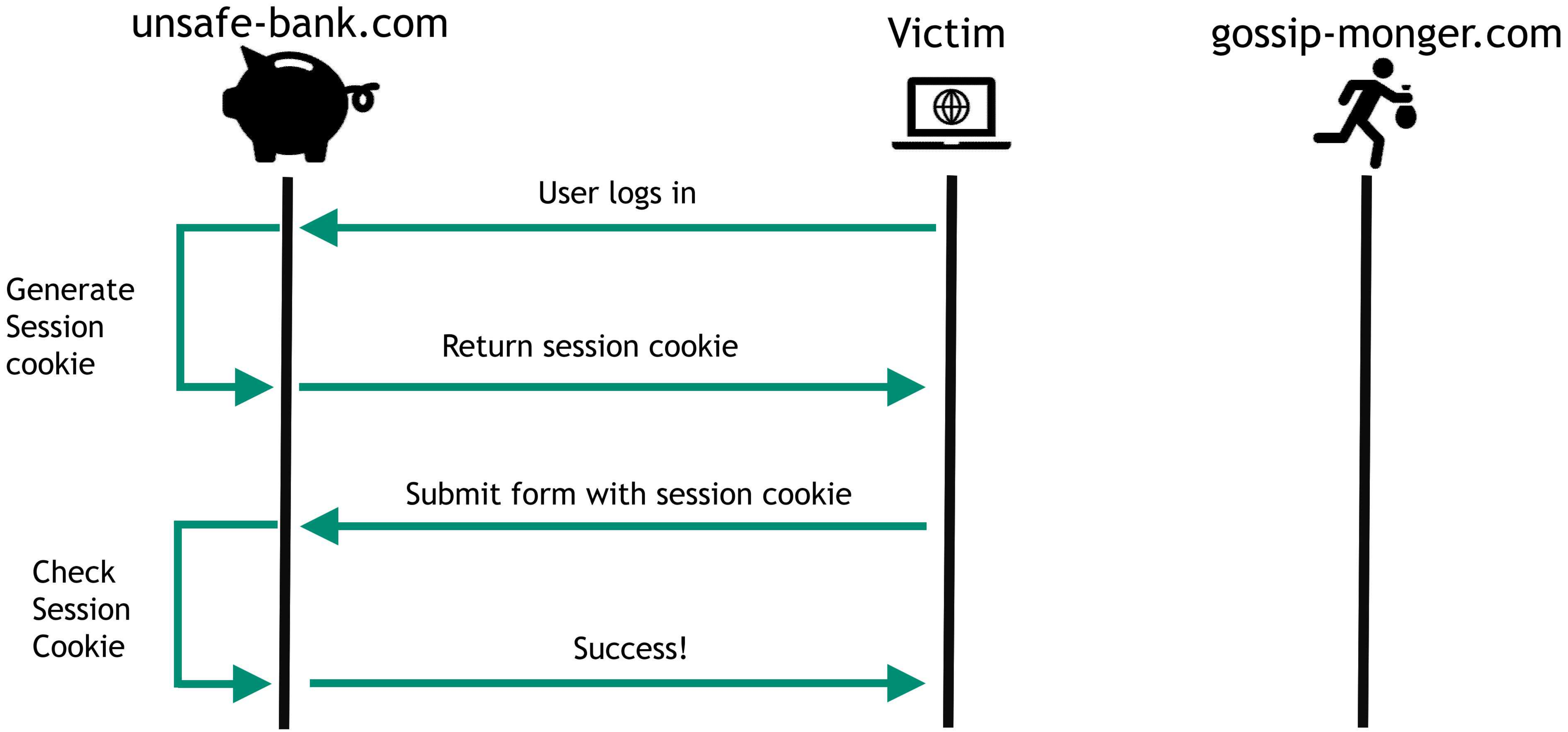
Pros

- No server state (i.e. session) is required
- Works with Single Page Applications
- Supported by many web frameworks
  - It is the default for Spring Security + Spring Boot

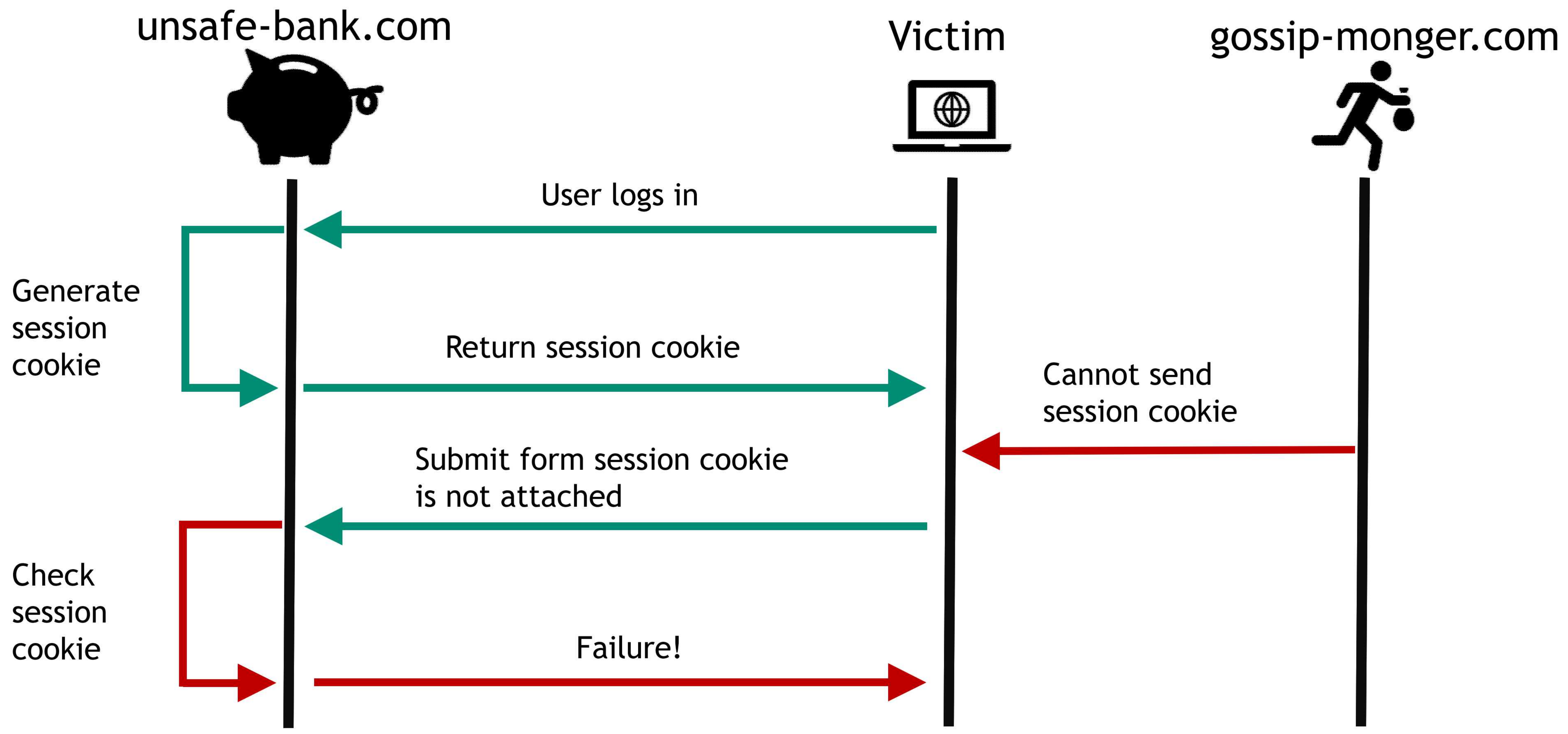
Cons

- Lifetime of the CSRF token is longer, providing more time for an attacker to subvert
- Vulnerable if the attacker can set the CSRF cookie to a known value
  - Possible if another site within the same domain (e.g. chatroom.unsafe-bank.com) is vulnerable

CSRF Mitigation: Same-Site Cookie



CSRF Mitigation: Same-Site Cookie



CSRF Mitigation: Same-Site Cookie

Pros

- Works with all applications, without changing any code
- Can configure cookies to be either strict or lax
  - Strict means the cookie will never be sent with cross-origin requests
  - Lax cross-origin allows GET and other "safe" HTTP methods are allowed

Cons

- Not supported by older browsers[1]
- May conflict with cookie-based single-sign on solutions
- In strict mode, clicking a link in a browser will likely return 'Page Not Found' if the user is already logged in
- Newer solution, so not as well supported by existing web frameworks
  - Not natively supported in Spring Security, for example, unlike the other CSRF mitigations

1. https://caniuse.com/#search=samesite, retrieved April 24, 2019

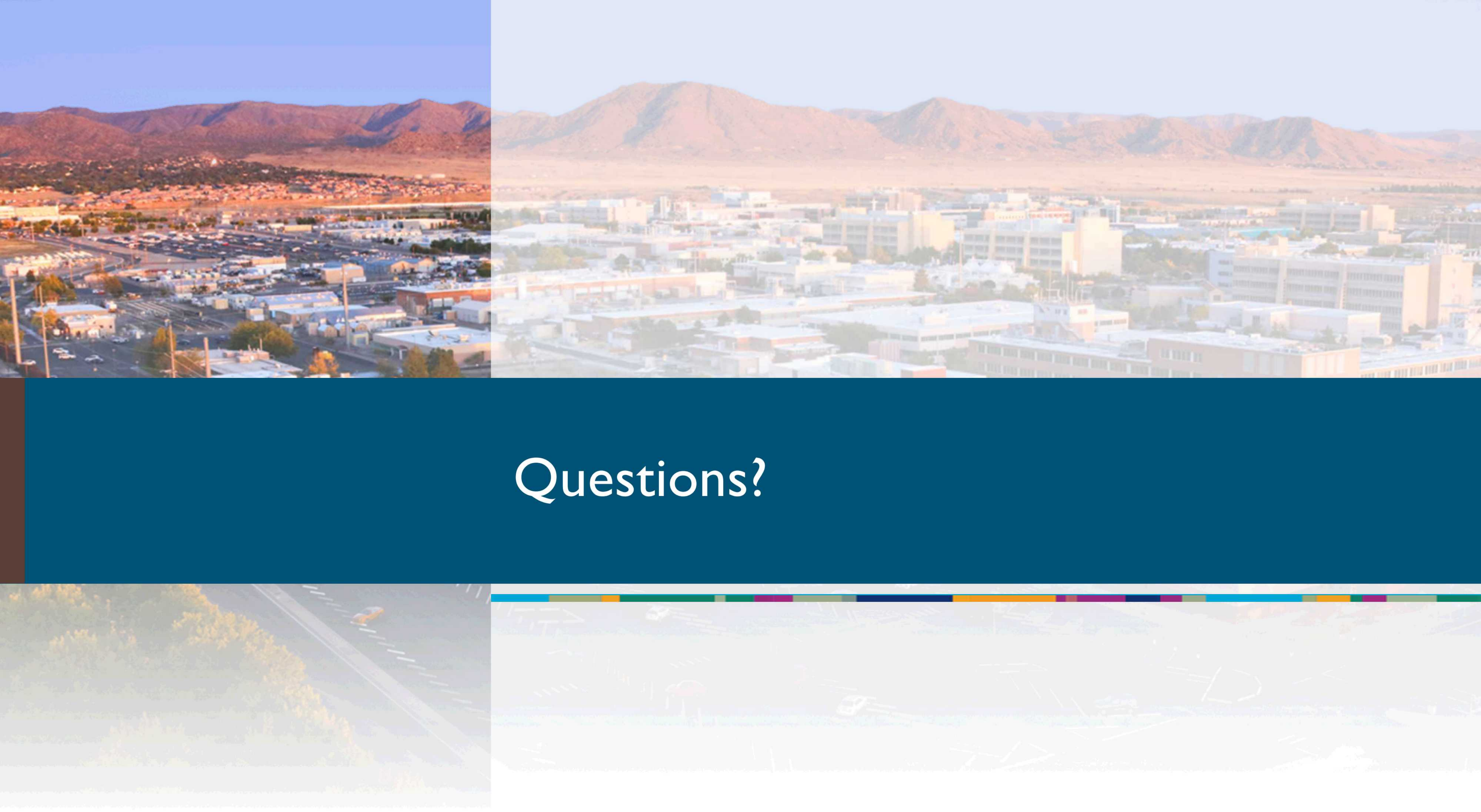
Summary

Cross Site Request Forgery (CSRF) attacks are real and still happening

They can be mitigated with common web frameworks providing built-in solutions

Providing details like this to development communities can help keep them thinking about security

Security can be hard, so having a specialized Software Security Group can be of benefit



Questions?