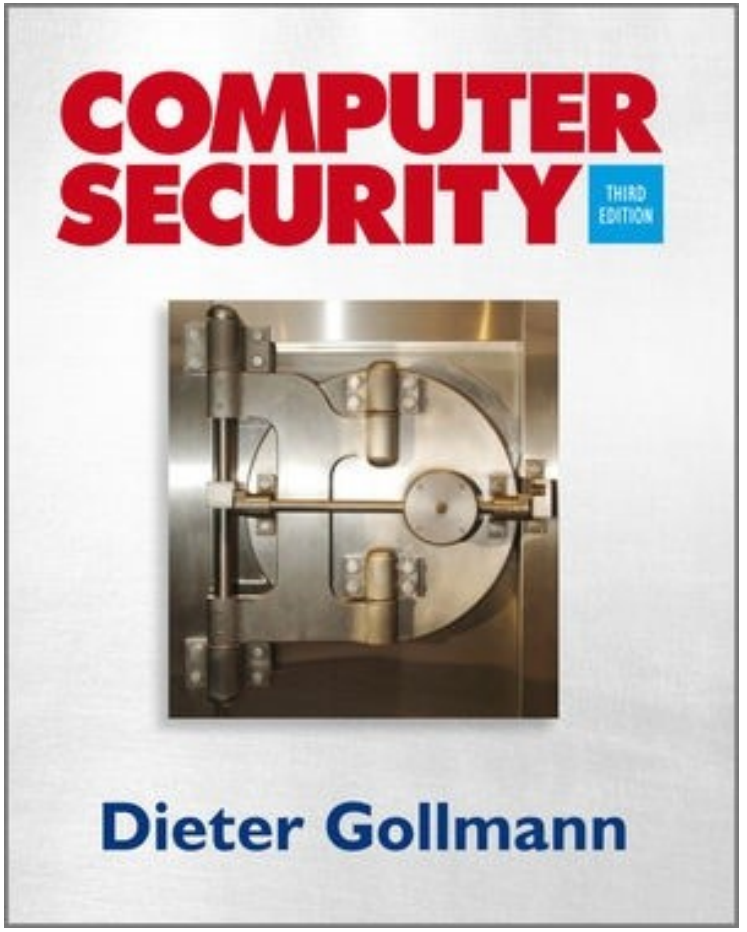
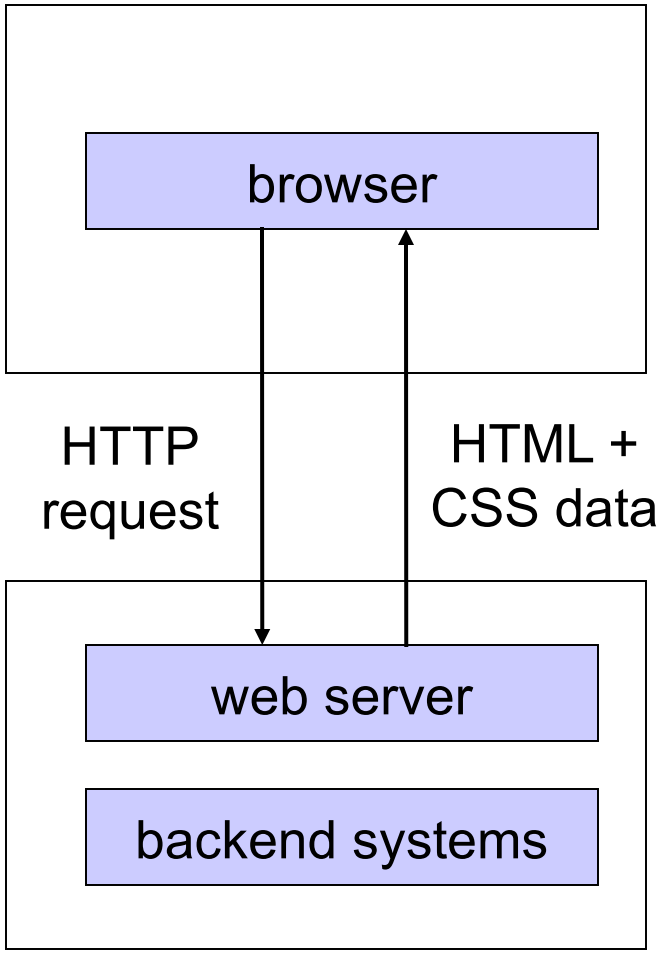
Computer Security 3e

Dieter Gollmann



Chapter 18:
Web Security

Web 1.0


browser
HTTP
request
HTML +
CSS data
web server
backend systems

Web 1.0

- Shorthand for web applications that deliver static content.
- At the client-side interaction with the application is handled by the browser.
- At the server-side, a web server receives the client requests.
- Scripts at web server extract input from client data and construct requests to a back-end server, e.g. a database server.
- Web server receives result from backend server; returns HTML result pages to client.

Transport Protocol

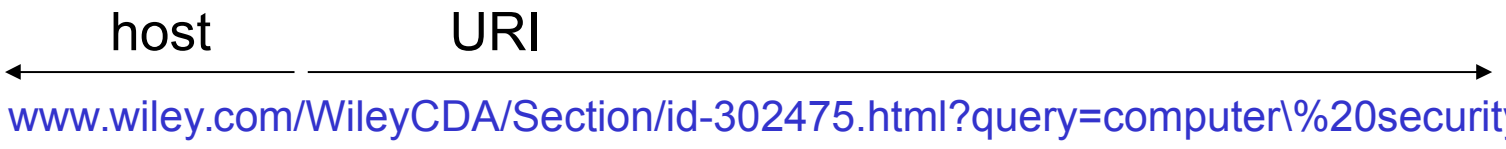
- Transport protocol used between client and server: HTTP (hypertext transfer protocol); HTTP/1.1 is specified in RFC 2616.
- HTTP located in the application layer of the Internet protocol stack.
- Do not confuse the network application layer with the business application layer in the software stack.
- Client sends HTTP requests to server.
- A request states a method to be performed on a resource held at the server.

HTTP GET & POST method

- GET method retrieves information from a server.
- Resource given by Request-URI (Uniform Resource Identifier) and Host fields in the request header.
- POST method specifies the resource in the Request-URI and puts the action to be performed on into the body of the HTTP request.
- POST was intended for posting messages, annotating resources, and sending large data volumes that would not fit into the Request-URI.
- In principle POST can be used for any other actions that can be requested by using the GET method but side effects may differ.

URI

- Parsing URI and Host:



- Attack: create host name that contains a character that looks like a slash; a user parsing the browser bar will take the string to the left of this character as the host name; the actual delimiter used by the browser is too far out to the right to be seen by the user.
- Defences:
  - Block dangerous characters.
  - Display to the user where the browser splits host name from URI; aligns the user's view with the browser's view.

HTML

- Server sends HTTP responses to the client. Web pages in a response are written in HTML (HyperText Markup Language).
- Elements that can appear in a web page include frame (subwindow), iframe (in-lined subwindow), img (embedded image), applet (Java applet), form.
- Form: interactive element specifying an action to be performed on a resource when triggered by a particular event; onclick is such an event.
- Cascading Style Sheets (CSS) for giving further information on how to display the web page.

Web Browser

- Client browser performs several functions.
  - Display web pages: the Document Object Model (DOM) is an internal representation of a web page used by browsers; required by JavaScript.
  - Manage sessions.
  - Perform access control when executing scripts in a web page.
- When the browser receives an HTML page it parses the HTML into the document.body of the DOM.
- Objects like document.URL, document.location, and document.referrer get their values according to the browser's view of the current page.

Web Adversary

- We do not assume the standard threat model of communications security where the attacker is “in control of the network” nor the standard threat model of operating system security where the attacker has access to the operating system command line.
- The web adversary is a malicious end system; this attacker only sees messages addressed to him and data obtained from compromised end systems accessed via the browser; the attacker can also guess predictable fields in unseen messages.
- The network is “secure”; end systems may be malicious or may be compromised via the browser.

Authenticated Sessions

- When application resources are subject to access control, the user at the client has to be authenticated as the originator of requests.
- Achieved by establishing an authenticated session.
- Authenticated sessions at three conceptual layers:
  - business application layer, as a relationship between user (subscriber) and service provider.
  - network application layer, between browser and web server.
  - transport layer, between client and server.
- TLS for authenticated sessions at the transport layer:
  - For users possessing a certificate and a corresponding private key, TLS with mutual authentication can be used.
  - EAP-TTLS when user and server share a password.

Session Identifiers

- Session identifier (SID): at the network application layer, created by the server and transmitted to client.
- In our threat model the SID can be captured once it is stored in an end system but not during transit.
- Client includes SID in subsequent requests to server; requests are authenticated as belonging to a session if they contain the correct SID.
- Server may have authenticated the user before the SID had been issued and encode this fact in the SID.
- Server may have issued the SID without prior user authentication and just use it for checking that requests belong to the same session.

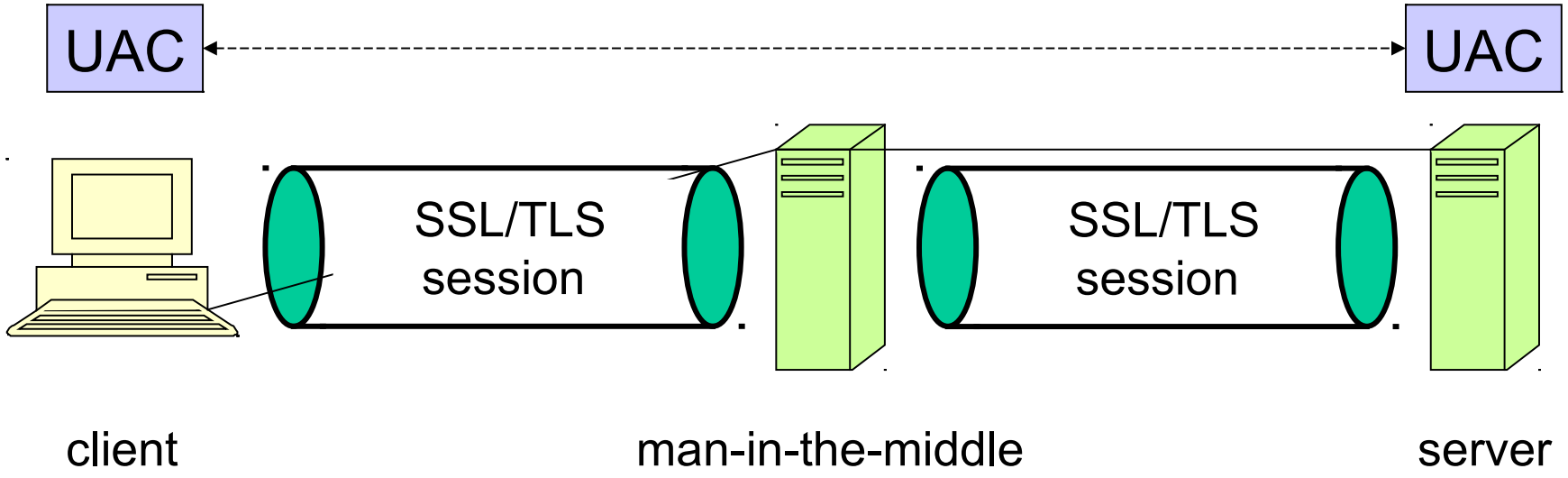
Transferring Session Identifiers

- Cookie: sent by server in a Set-Cookie header field in the HTTP response; browser stores cookie in document.cookie and includes it in requests with a domain matching the cookie's origin.
- URI query string: SID included in Request-URIs.
- POST parameter: SID stored in a hidden field in an HTML form.
- At the business application layer, the server can send an authenticator to the client; client has to store authenticator in the private space of the application.

Cookie Poisoning

- If SIDs are used for access control, malicious clients and outside attackers may try to elevate their permissions by modifying a SID (cookie).
- Such attacks are known as cookie poisoning.
- Outside attackers may try educated guesses about a client's cookie, maybe after having contacted the server themselves.
- Attacker may try to steal cookie from client or server.
- Two requirements on session identifiers: they must be unpredictable; they must be stored in a safe place.
- Server can prevent modification of SID by embedding a cryptographic message authentication code in the SID constructed from a secret only held at the server.

Man-in-the-Middle attack



Is the user authenticator UAC (better: request authenticator) bound to SSL/TLS session?

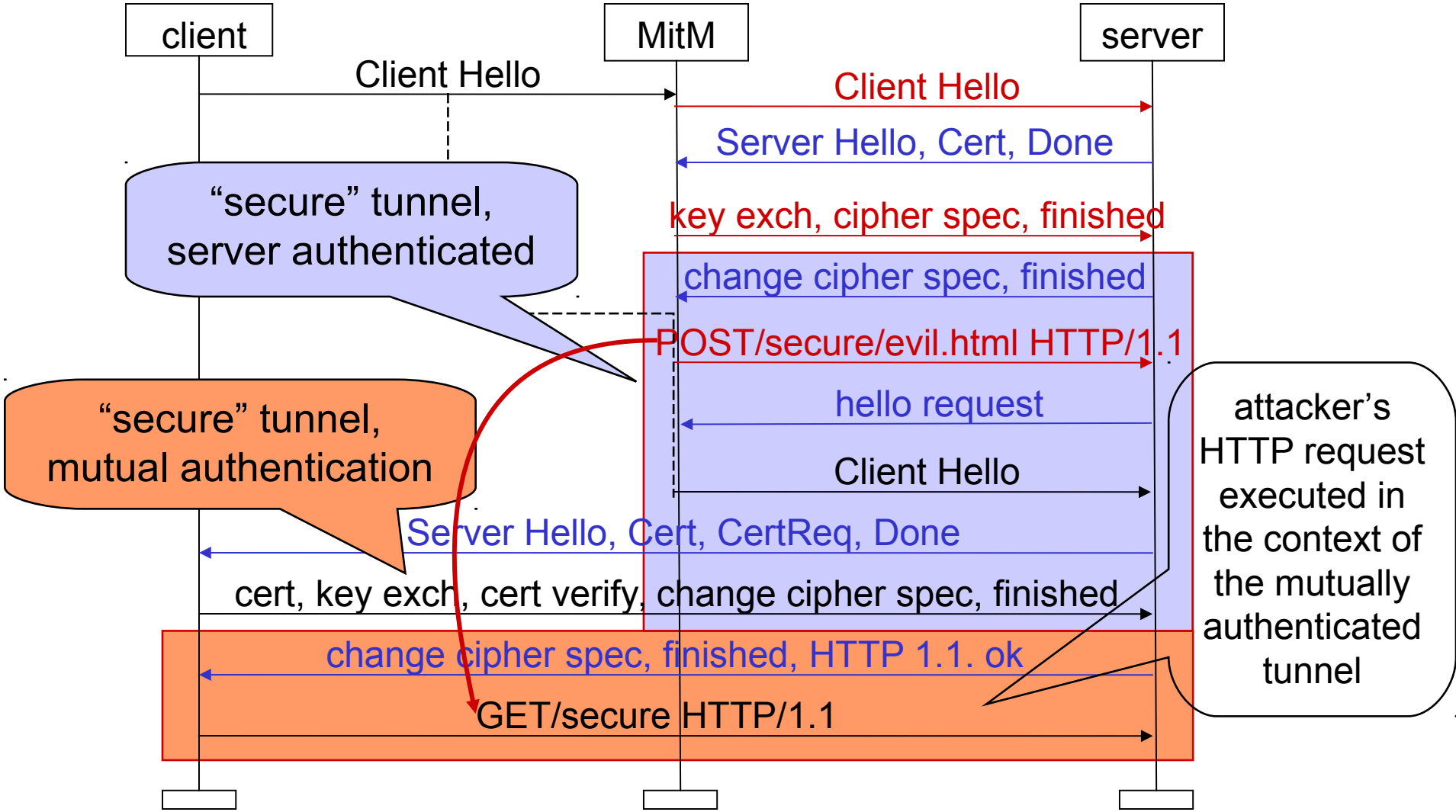
Session-Aware User Authentication

- Authenticate requests in browser session:
  - Client establishes SSL/TLS session to server.
  - Sends user credentials (e.g. password) in this session.
  - Server returns user authenticator (e.g. cookie); authenticator included by client in further HTTP requests.
- Bind authenticator not only to user credentials but also to SSL/TLS session in which credentials are transferred to server.
- Server can detect whether requests are sent in original SSL/TLS session.
  - If this is the case, probably no MiTM is involved.
  - If a different session is used, it is likely that a MiTM is located between client and server.

Recent TLS Security Scare

- “Flaw” of TLS widely reported.
  - Marsh Ray, Steve Dispensa: Renegotiating TLS, 4.11.2009
- Background: TLS employed for user authentication when accessing a secure web site.
- Common practice for web servers to let users start with an anonymous TLS session.
- Request for a protected resource triggers TLS renegotiation; mutual authentication requested when establishing the new TLS tunnel.

https-Problem



Comment

- Web developers using session renegotiation for user authentication assumed features not found in RFC 5246.
- Fact: typical use case for renegotiation suggests that the new session is a continuation of the old session.
  - Plausible assumptions about a plausible use case are treated as a specification of the service.
- Fix: TLS renegotiation cryptographically tied to the TLS connection it is performed in (RFC 5746).
  - TLS adapted to meet expectations of application.
- This had really been an application layer problem.
  - State at server persists over two TLS tunnels; attacker sends a malicious partially complete command in the first tunnel.

Same Origin Policy

Same Origin Policy

- Web applications can establish sessions (common state) between participants and refer to this common state when authorising requests.
- Sessions between client and server established through cookies, session identifiers, or SSL/TLS.
- Same origin policies enforced by web browsers to protect application payloads and session identifiers from outside attackers.
  - Script may only connect back to domain it came from.
  - Include cookie only in requests to domain that had placed it.
- Two pages have the same origin if they share the protocol, host name and port number.

Evaluating same origin for http://www.my.org/dir1/hello.html

URL	Result	Reason
http://www.my.org/dir1/some.html	success	
http://www.my.org/dir2/sub/another.html	success	
https://www.my.org/dir2/some.html	failure	different protocol
http://www.my.org:81/dir2/some.html	failure	different port
http://host.my.org/dir2/some.html	failure	different host

Same Origin Policy: Exceptions

- Web page may contain images from other domains.
- Same origin policy is too restrictive if hosts in same domain should be able to interact.
- Parent domain traversal: Domain name may be shortened to its .domain.tld portion.
  - www.my.org can be shortened to my.org but not to .org.
- Undesirable side effects when DNS is used creatively.
  - E.g., domain names of UK universities end with .ac.uk.
  - ac.uk is no proper Top Level Domain.
  - Restricting access to domain.tld portion of host name leaves all ac.uk domains open to same origin policy violations.
  - Browsers are shipped with a list of domains where parent domain traversal cannot be applied (exceptions to exception).

Same Origin Policy: Variants

- Same origin policy for HTML cookies requires host+path to be the same.
- JavaScript same origin policy on document.cookies in the DOM considers host+protocol+port.
- Internet Explorer does not consider the port when evaluating the same origin policy.
- For https, it may be advisable to include the session key in the same origin policy.
  - Prevents interference between different “secure” sessions to the same server.

Cross Site Scripting

Cross Site Scripting – XSS

- Parties involved: attacker, client (victim), server ('trusted' by client).
  - Trust: code in pages from server executed with higher privileges at client (origin based access control).
- Attacker places malicious script on a page at server (stored XSS) or gets victim to include attacker's script in a request to the server (reflected XSS).
- If the script contained in page is returned by the server to the client in a result page, it will be executed at client with permissions of the trusted server.
- Evades client's origin based security policy

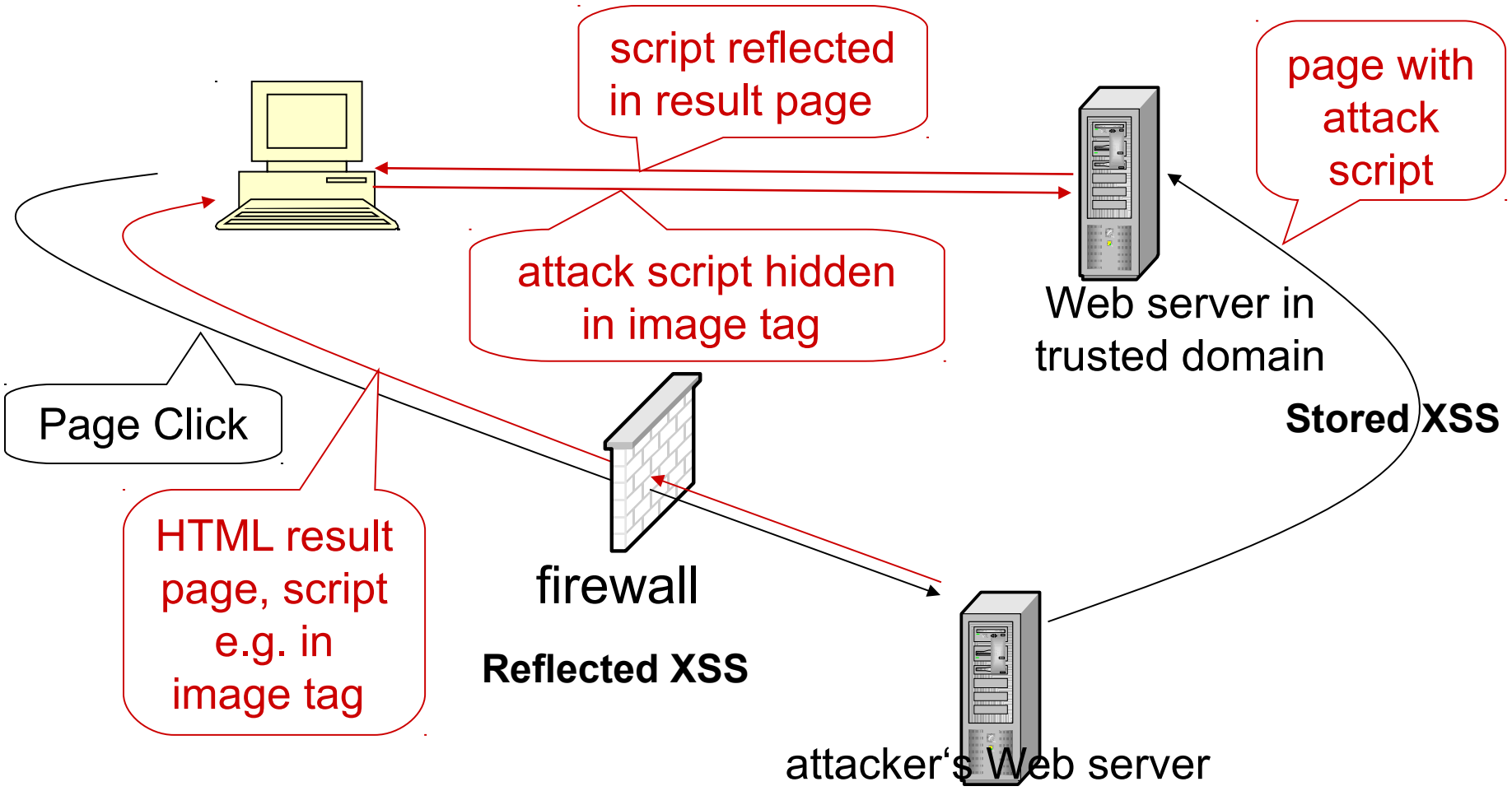
Reflected XSS

- Data provided by client is used by server-side scripts to generate results page for user.
- User tricked to click on attacker's page for attack to be launched; page contains a frame that requests page from server with script as query parameter.
- If unvalidated user data is echoed in results page (without HTML encoding), code can be injected into this page.
- Typical examples: search forms, custom 404 pages (page not found)
  - E.g., search engine redisplays search string on the result page; in a search for a string that includes some HTML special characters code may be injected.

Stored XSS

- Stored, persistent, or second-order XSS.
- Data provided by user to a web application is stored persistently on server (in database, file system, …) and later displayed to users in a web page.
- Typical example: online message boards.
- Attacker places a page containing malicious script on server.
- Every time the vulnerable web page is visited, the malicious script gets executed.
- Attacker needs to inject script just once.

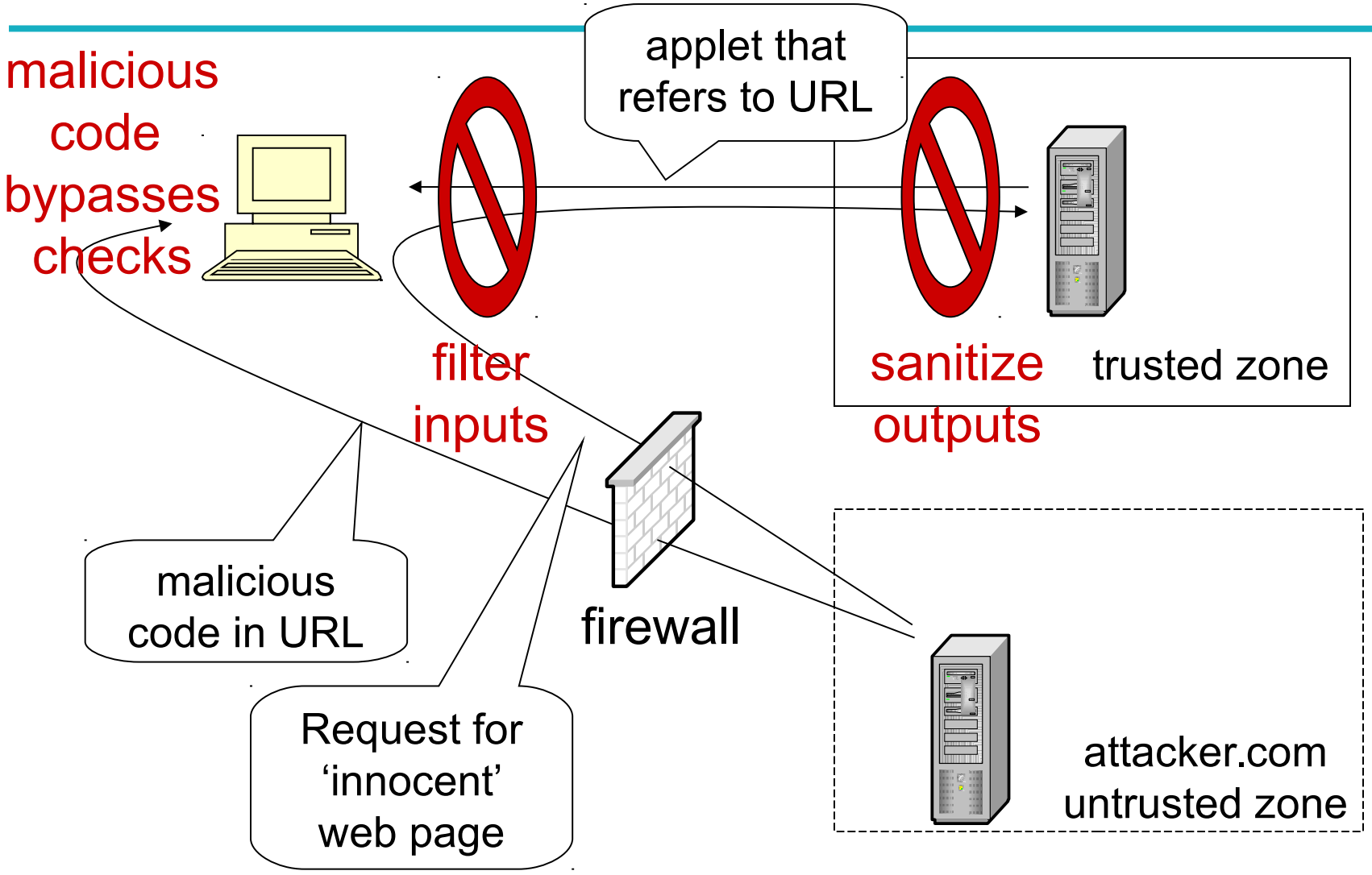
Cross-site Scripting



DOM-based XSS

- HTML parsed into document.body of the DOM.
- document.URL, document.location, document.referrer assigned according to browser's view of current page.
- Scripts in a web page may refer to these objects.
- Attacker creates page with malicious script in the URL and a request for a frame on a trusted site; result page contains script that references document.URL.
- User clicks on link to this page; browser puts bad URL in document.URL, requests frame from trusted site.
- Script in results page references document.URL; now the attacker's code will be executed.

DOM-based XSS



Threats

- Execution of code on the victim's machine.
- Cookie stealing & cookie poisoning: read or modify victim's cookies.
  - Attacker's script reads cookie from document.cookie, sends its value back to attacker, e.g. as HTTP GET parameter.
  - No violation of the same origin policy as script runs in the context of attacker's web page.
- Execute code in another security zone.
- Execute transactions on another web site (on behalf of a user).
- Compromise a domain by using malicious code to refer to internal web pages.

XSS – The Problem

- Ultimate cause of the attack: The client only authenticates ‘the last hop’ of the entire page, but not the true origin of all parts of the page.
- For example, the browser authenticates the bulletin board service but not the user who had placed a particular entry.
- If the browser cannot authenticate the origin of all its inputs, it cannot enforce a code origin policy.

Defences

- Three fundamental defence strategies:
- Change modus operandi: block execution of scripts in the browser; e.g., block in-line scripts.
- Abandon the same origin policy; try to differentiate between code and data instead.
  - Clients can filter inputs, sanitize server outputs, escape, encode dangerous characters.
- Authenticate origin (without relying on a PKI).

Cross site request forgery

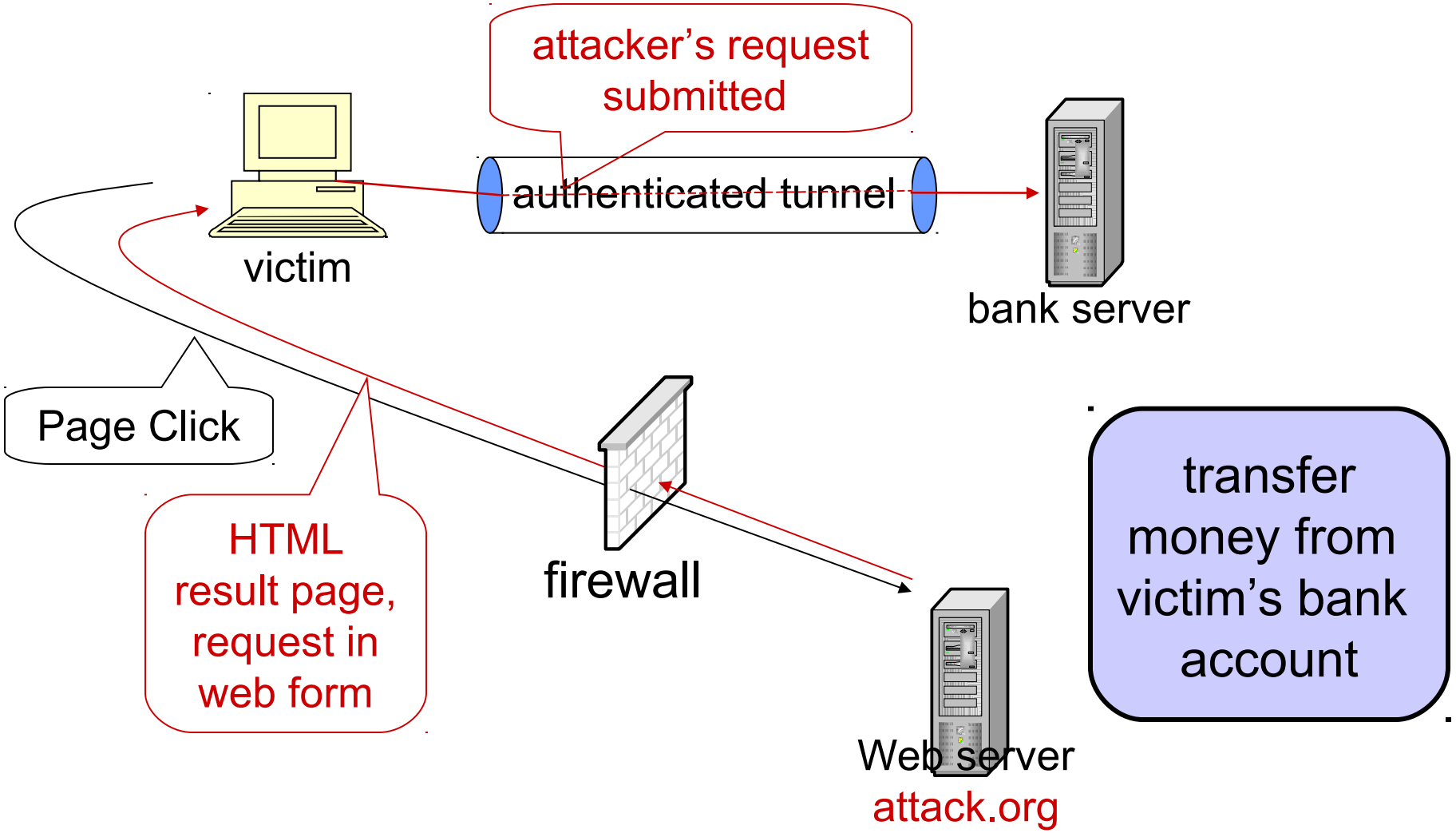
XSRF Attack

- Parties involved: attacker, user, target web site.
- Cross-site request forgery (XSRF) exploits 'trust' a website has in a user to execute malware at a target website with the user's privileges.
  - Trust: user is somehow authenticated at the target website (cookie, authenticated session,…).
- User has to visit a page placed by the attacker, which contains hidden request, e.g. in an HTML form.
- Evades target's origin based security policy.

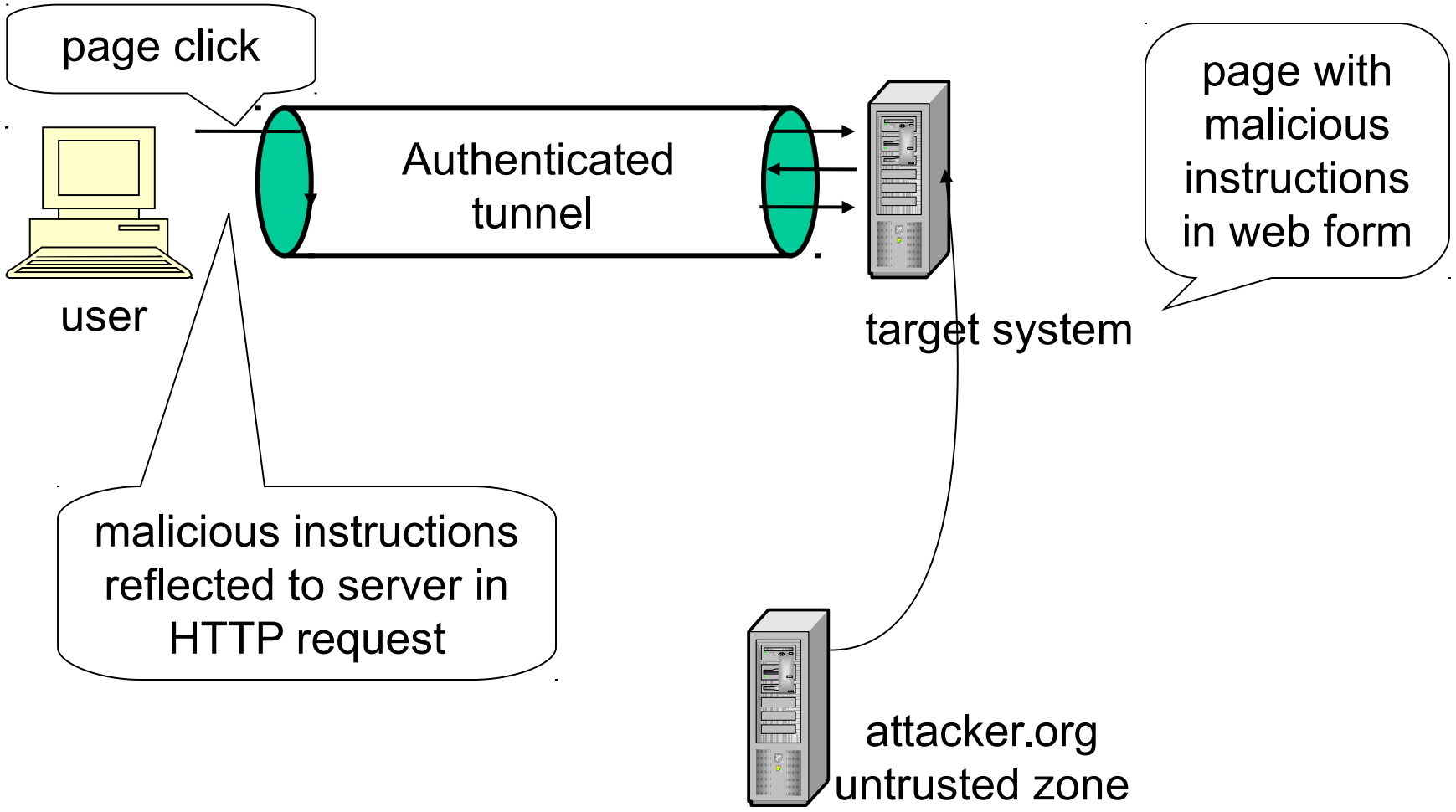
XSRF Attack

- Reflected XSRF (user has to visit page at attacker's site) and stored XSRF (attacker places page at server).
- When the user browses this page, JavaScript automatically submits the form data to the target site where the user has access.
- Target authenticates request as coming from user; form data accepted by server since it comes from a legitimate user.

Reflected XSRF



Stored XSRF


page click
Authenticated tunnel
page with malicious instructions in web form
user
target system
malicious instructions reflected to server in HTTP request
attacker.org
untrusted zone