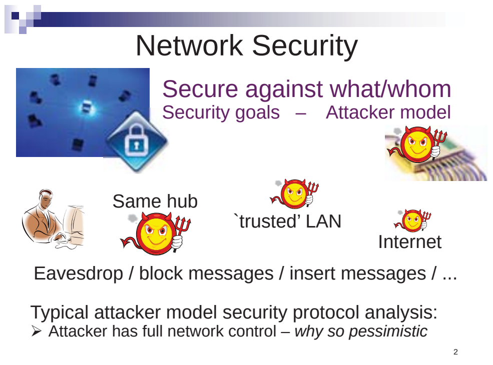
3 Network and Web security

3.1 Network Security

When different systems are interconnected, the problem of security grows significantly. Not only do we need to consider threats on our own system but also on all systems connected to it, as well as the connections themselves. In this section we will look at some of these threats and related countermeasures.

Of key importance when considering the threats and attacks described in this section is to keep in mind who and where the attacker is and what she is trying to achieve; i.e. what is the **attacker model** that belongs to the threat. Different attackers will have **different capabilities**. For example, an attacker on the same hub will see all the messages being sent. An attacker on the same, trusted, local network (LAN) will be able to perform attacks without having to worry about an internet firewall protecting the LAN. Attacks by different attacks will also have **different goals**. An attack may, for example, be aimed at getting a message onto the LAN. Obviously such an attack is only relevent for an outside attacker as an inside attacker already has this capability.


Network Security

Secure against what/whom

Security goals – Attacker model

Same hub

`trusted' LAN

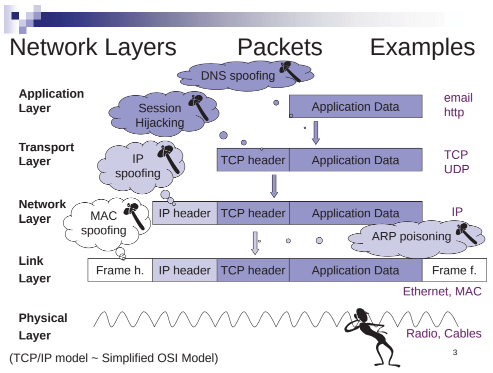
Internet

Eavesdrop / block messages / insert messages / ...

Typical attacker model security protocol analysis:

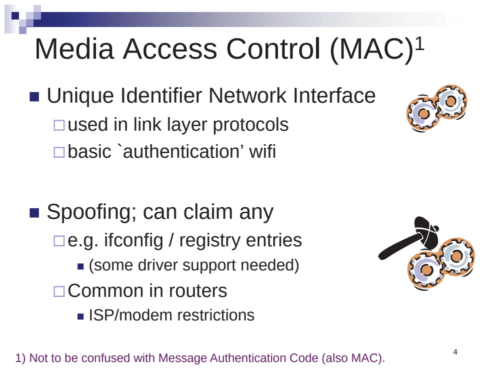
➢ Attacker has full network control – *why so pessimistic*

2



To secure the network we need to considering attacks at different layers. Consider the network layer model for TCP/IP. An application will get the transport layer to manage a connection with a party it wants to communicate with. Human understandable addresses such as `www.tue.nl` or `somebody@tue.nl` need to be translated to IP addresses understood by the network layer. An attacker may try to get the traffic redirected to their IP by disturbing this step (e.g. through DNS spoofing). The attacker could also influence lower layers to achieve the same result. For example an attacker could eavesdrop messages if she has access to the physical layer.

MAC The media access control (MAC) protocol is used on the link layer. A MAC address is used to identify each network device on the network. Some wireless routers use MAC addresses white-listing as a security mechanism, allowing only access by the listed MAC addresses. However, a network device can claim to be have any MAC address. Some routers even support setting the MAC address that they use as some internet modems only talk to one fixed MAC address. Inserting a new router would then not be possible.

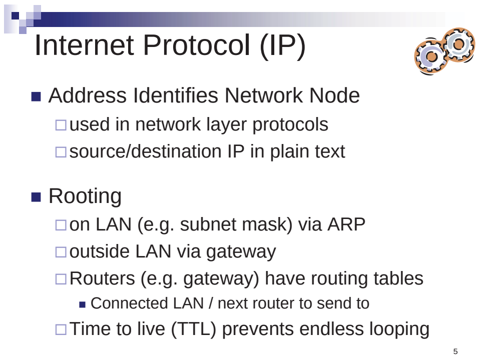


Media Access Control (MAC)[1]

- Unique Identifier Network Interface
  - used in link layer protocols
  - basic `authentication' wifi
- Spoofing; can claim any
  - e.g. ifconfig / registry entries
    - (some driver support needed)
  - Common in routers
    - ISP/modem restrictions

1) Not to be confused with Message Authentication Code (also MAC).

4

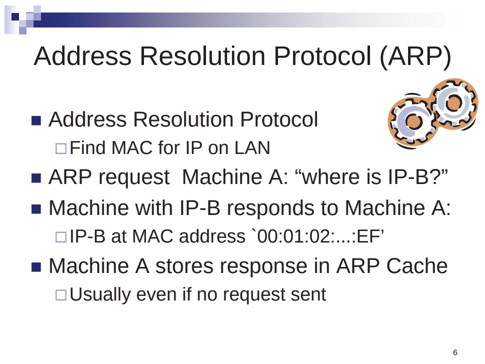


Internet Protocol (IP)

- Address Identifies Network Node
  - used in network layer protocols
  - source/destination IP in plain text
- Rooting
  - on LAN (e.g. subnet mask) via ARP
  - outside LAN via gateway
  - Routers (e.g. gateway) have routing tables
    - Connected LAN / next router to send to
  - Time to live (TTL) prevents endless looping

5

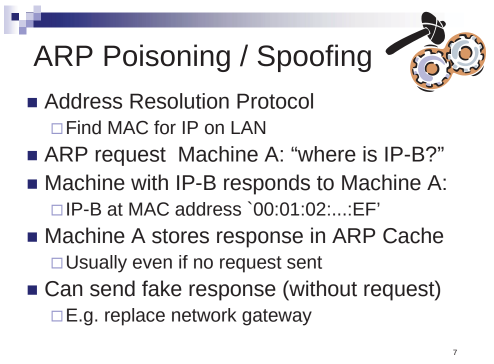
ARP: IP to MAC On the network layer IP address are used to identify devices on the network. Routing of IPs depends on whether the IP is local or nor (as configured by the subnet mask). Local addresses are sent to the MAC address of the machine that belongs to that IP. Other addressed are sent to the gateway responsible for passing messages to the rest of the network. If machine A does not have the MAC address that belongs to some local IP-B in its cache it will use the address resolution protocol (ARP) to broadcast the question 'where is IP-B?' the machine with IP-B will respond by sending 'IP-B is at MAC x'. For efficiency machine A typically does not remember whether or not it asked for IP-B. Instead, whenever it sees the message 'IP-B is at MAC x' it updates it cache. Thus to claim an IP-B all that is needed is to send a fake response (even without there needing to be a request).



Address Resolution Protocol (ARP)

- Address Resolution Protocol
  - Find MAC for IP on LAN
- ARP request Machine A: "where is IP-B?"
- Machine with IP-B responds to Machine A:
  - IP-B at MAC address `00:01:02:...:EF'
- Machine A stores response in ARP Cache
  - Usually even if no request sent

6



ARP Poisoning / Spoofing

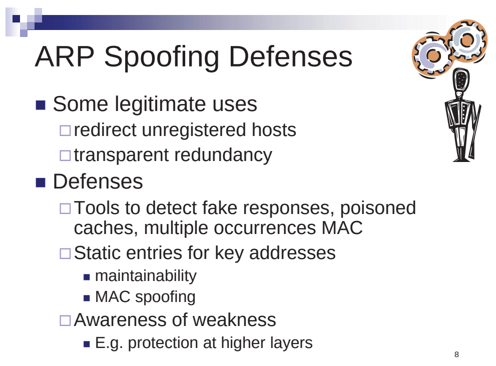
- Address Resolution Protocol
  - Find MAC for IP on LAN
- ARP request Machine A: "where is IP-B?"
- Machine with IP-B responds to Machine A:
  - IP-B at MAC address `00:01:02:...:EF'
- Machine A stores response in ARP Cache
  - Usually even if no request sent
- Can send fake response (without request)
  - E.g. replace network gateway

7

So what can be done to prevent ARP spoofing? One thing to note is that there are some legitimate uses possible where the redirection of traffic here is actually intentional, not an attack. For example, a backup server may transparently take over the role of crashed server by claiming its IP. Other uses may be to redirect a new machine on the network to a sign-up page before giving it access e.g. to (the gateway and) the internet.

Tools which monitor the network can be used to look for fake responses (e.g. responses without requests, multiple responses to a request), poisoned ARP caches (e.g. different values for the same IP, does not match confirmed values), etc. To limit possible damage one could also use static entries for key addresses (such as the gateway, important local services) and not use ARP for these addresses. A disadvantage here is the maintenance involved; if any of they static addresses change, all devices on the network have to be updated. Also, recall that any device can claim to have any MAC address so using the correct MAC address does not guarantee that the message goes to the correct machine.

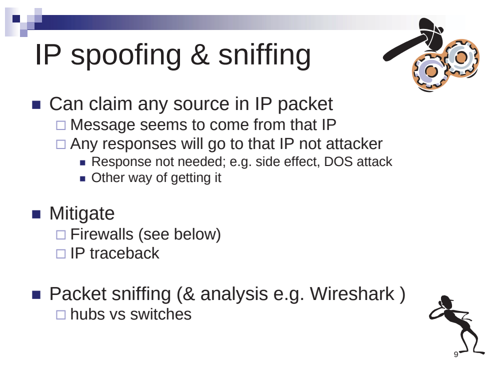
Instead of trying to prevent the ARP spoofing we can try to solve this at the higher network layers, taking into account the fact that the lower layer may be unreliable in the design of protocols for the higher layers.



ARP Spoofing Defenses

- Some legitimate uses
  - □ redirect unregistered hosts
  - □ transparent redundancy
- Defenses
  - □ Tools to detect fake responses, poisoned caches, multiple occurrences MAC
  - □ Static entries for key addresses
    - maintainability
    - MAC spoofing
  - □ Awareness of weakness
    - E.g. protection at higher layers

8



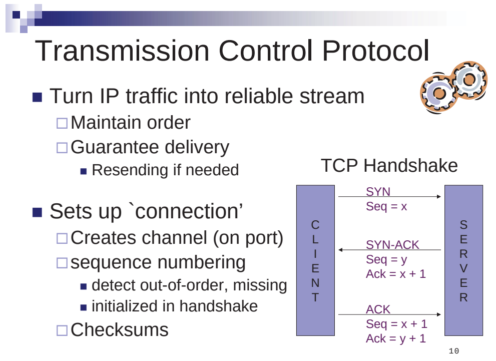
IP spoofing & sniffing

- Can claim any source in IP packet
  - □ Message seems to come from that IP
  - □ Any responses will go to that IP not attacker
    - Response not needed; e.g. side effect, DOS attack
    - Other way of getting it
- Mitigate
  - □ Firewalls (see below)
  - □ IP traceback
- Packet sniffing (& analysis e.g. Wireshark)
  - □ hubs vs switches

9

IP Like the MAC address, the IP address is just a plain text string inside the message. Spoofing an IP address in a message is simple; just change the IP header of the message. The message will then appear to come from (or be intended for if you change the destination) that IP address.

To help mitigate attacks based on IP spoofing one can use firewalls to block message from outside the network that claim to come from an IP on the local network. Also, it may be possible to trace back the source of the message (the routers forwarding the message may allow) which may help finding the source of the attack, or at least defend against it closer to its source (which may be important e.g. in denial of service attacks). Note that the attacker model has changes compared to the discussion on e.g. MAC. With MAC spoofing we consider an attacker on the local network. IP spoofing is basically possible for any attacker. The firewall defense clearly only makes sense when we are dealing with an outside attacker. (As we have seen above an attacker on the local network can likely mount a more powerful attack, actually claiming the IP rather than only doing IP spoofing.)



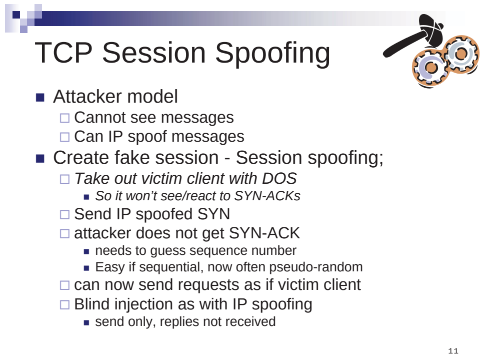
Transmission Control Protocol

- Turn IP traffic into reliable stream
  - □ Maintain order
  - □ Guarantee delivery
    - Resending if needed
- Sets up `connection'
  - □ Creates channel (on port)
  - □ sequence numbering
    - detect out-of-order, missing
    - initialized in handshake
  - □ Checksums

TCP Handshake

CLIENT
SERVER
SYN
Seq = x
SYN-ACK
Seq = y
Ack = x + 1
ACK
Seq = x + 1
Ack = y + 1

10



TCP Session Spoofing

- Attacker model
  - □ Cannot see messages
  - □ Can IP spoof messages
- Create fake session - Session spoofing;
  - □ *Take out victim client with DOS*
    - *So it won't see/react to SYN-ACKs*
  - □ Send IP spoofed SYN
  - □ attacker does not get SYN-ACK
    - needs to guess sequence number
    - Easy if sequential, now often pseudo-random
  - □ can now send requests as if victim client
  - □ Blind injection as with IP spoofing
    - send only, replies not received

11

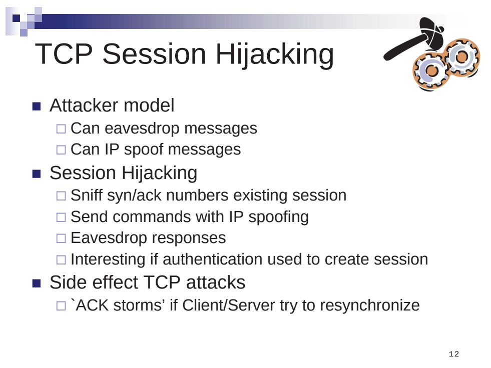
TCP Network traffic is not by itself reliable; packets may get lost along the way or overtaken by newer packets. The TCP protocol aims to ensures that message are delivered completely and in the right order. The protocol uses sequence numbers to identify blocks of communication and these sequence numbers form a basic form of authentication; packets are only accepted if the right sequence number is used.

If we can predict (see e.g. [3] and compare with lab session 6 on http session stealing) the (initial) sequence number that the server will use we can do a blind injection with a spoofed IP and guessed sequence number to start our own session with the server; we don't see the responses but we can issue request which will be accepted as coming from the (trusted) client. We do need to prevent the real client from reacting to messages as seeing unexpected session number from the server will cause it to send a reset request which would break our connection. Note that this attack is more a fake session initiation than a session hijacking attack but is typically seen as belonging to the category of session hijacking attacks.

Clearly if we can somehow get the communication between the server we can simply read the messages and the sequence numbers within them. (We could for example pretend to be a gateway between the server and the client networks with the techniques described above or we could also try to get responses from the server to be sent back to use by using source routing; in which the sender of the packet can indicate (part of) the route that the return packet should follow, however source routing is now usually disabled.)

A session between a client and server may start with authentication of the client. In this case we may want to actively take over an (authenticated) session rather than try to start our own new one. If we can see the traffic between the server we can wait for a connection to be created and authentication to complete, eavesdrop the session numbers used and then inject our own messages with IP spoofing using the eavesdropped session numbers. Variants of this attack differ in how they deal with the client and server responses. For example we can take down the client after the session has started to prevent it from interfering in our stolen session. We could also desynchronize the client and server by resetting the connection on one end but not the other; we now act as a man in the middle forwarding the responses of the client and server (adapting the session numbers) when we want but changing them as we please. If the session includes setting up encryption keys (see e.g. DH Key exchange treated earlier) then this type of a man in the middle attack can also break protection at higher network levels that rely on this encryption.

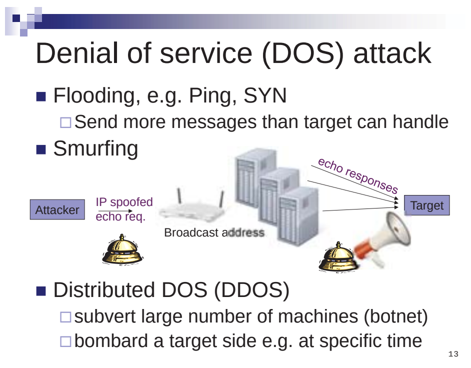
A problem with some variations of the attack can be so called 'ACK storms' which can give away the attack; the server acknowledges the data that we send to it but will of course send its acknowledgment packets to the client. If we don't prevent these from reaching the client it will, upon receiving these packets, notice that the acknowledgment number is incorrect and send an ACK packet back with the correct sequence number in an attempt to re-establish a synchronized state. However, this response itself will be seen as invalid by the server, leading to another ACK packet, creating a loop that continues until one of the ACK packets is lost in transit.



TCP Session Hijacking

- Attacker model
  - Can eavesdrop messages
  - Can IP spoof messages
- Session Hijacking
  - Sniff syn/ack numbers existing session
  - Send commands with IP spoofing
  - Eavesdrop responses
  - Interesting if authentication used to create session
- Side effect TCP attacks
  - `ACK storms' if Client/Server try to resynchronize

12



Denial of service (DOS) attack

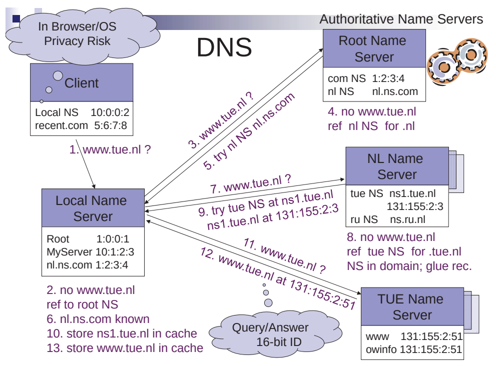
- Flooding, e.g. Ping, SYN
  - Send more messages than target can handle
- Smurfing

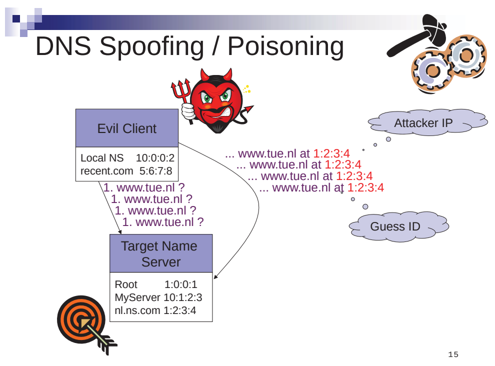
Attacker
IP spoofed echo req.
Broadcast address
echo responses
Target

- Distributed DOS (DDOS)
  - subvert large number of machines (botnet)
  - bombard a target side e.g. at specific time

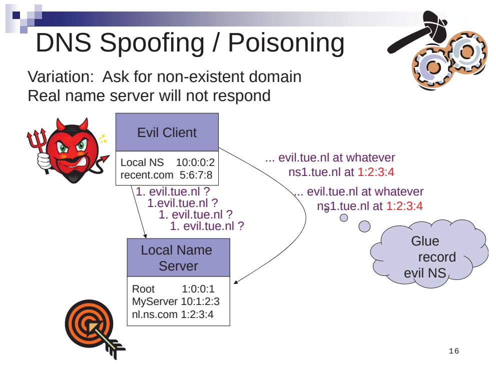
13

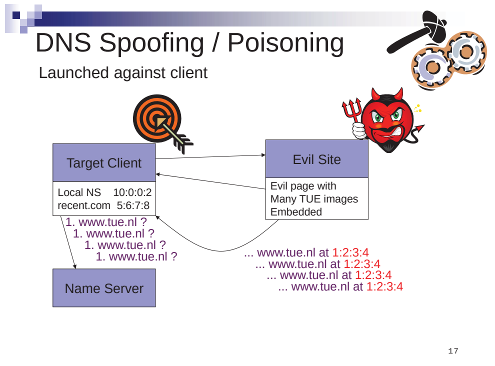
DNS: Domain(url) to IP The Domain Name System (DNS) is a key ingredient of the World Wide Web. It translates human readable addresses (domains), such as `www.tue.nl` to IP addresses. A client (e.g. a web browser) typically keeps a local cache (in the brower itself or in the OS running the browser) of the Domain-IP mapping. However, for new domains it will need to contact a name server. The local name server is typically set along with the IP address of the machine. The local name server has a link to a root name server and a cache of its own. If (1) the client tries to look up `www.tue.nl` and (2) this domain is not in the cache the name server will (3) ask the root name server. The root name server does not have (4) the IP for domain `www.tue.nl` itself but does know which name server is responsible for the top level domain; in this case e.g. `nl.ns.com` for the `.nl` domain, which it returns (5) to the local name server. Next (7) the local name server contacts this server and again gets redirected (8,9), now to `ns1.tue.nl`. As this name server is actually within the `tue.nl` domain, the IP for this domain is added to this answer (9) so we can actually contact this server to learn the (other) IPs in the `tue.nl`-domain. Finally we get the IP we are looking for (11,12) which is stored in the cache and returned to the client (not depicted). A 16-bit ID is used to link an answer to a query.





There are several types of attacks possible on the DNS scheme. In the first we look at the attacker is a client; that is trying to poison the cache of a name server. It sends a lot of requests for the address it wants to be linked to its own IP. If the address is not yet in the cache the name server will send out requests for this address. The attacker in the mean time sends many replies to this request containing its own IP and using different values for the ID. If one of the IDs used by the attacker matches one of the IDs used in the request of the name server the response will be accepted and the attackers address set in the cache of the name server. (Consider the attacker model; which implicit assumptions are made and why?)



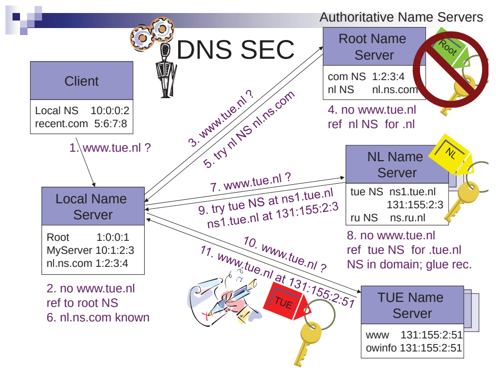


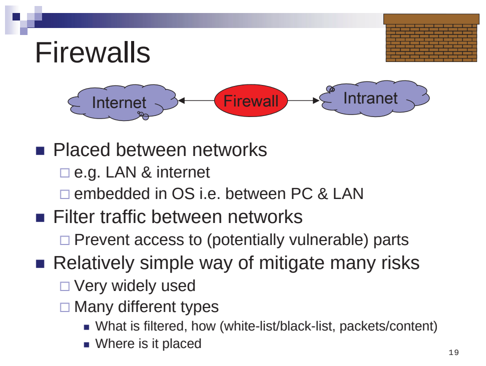
In a variation on the attack above, the attacker asks for a non-existent domain. The advantage is that the domain will for sure not be in the cache of the local name server and no name server will resolve the domain so the attacker has no competition for its response queries. But what use is getting the name server to have a false record for a non-existent domain? The clue is the use of glue records; if the attacker can get the name server to accept her response she can include a glue record which will also be stored and can make an important domain (e.g. that of a name server) point at her IP.

Not only the name server can come under attack. Also the client itself may be targeted. For example, an evil web site could contain many items from the `www.tue.nl` domain. If the client visits the page this will causing many requests for www.tue.nl and the attacker can send many responses at the same time, hoping to match the ID of one of the requests.

3.1.1 Network security technologies

One possible defense against the DNS attacks described above is to authenticate responses using digital signatures. In DNS SEC (Domain Name System Security Extensions) the name server can sign its response so the attacker cannot create a fake response. Of course we do need to be able to check that the public key of the name server is correct. The way to do this is to have the name server that give us this domain also have it validate its public key (See lecture on certificate and certificate authorities for more on this structure). Of course if any step in the chain does not deploy DNS SEC the system fails; it thus needs to be widely deployed before bringing real benefits. This and added complexity, key management issues and potential information leakage (e.g. which names exist in a domain) that could be caused by DNS SEC have hindered deployment, e.g. only since 2010 has DNS SEC been deployed at the root level at all.





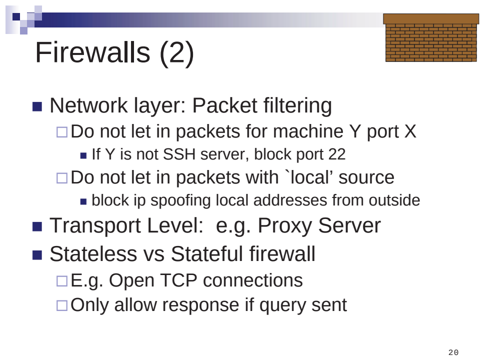
Firewalls

Internet → Firewall → Intranet

- Placed between networks
  - e.g. LAN & internet
  - embedded in OS i.e. between PC & LAN
- Filter traffic between networks
  - Prevent access to (potentially vulnerable) parts
- Relatively simple way of mitigate many risks
  - Very widely used
  - Many different types
    - What is filtered, how (white-list/black-list, packets/content)
    - Where is it placed

19

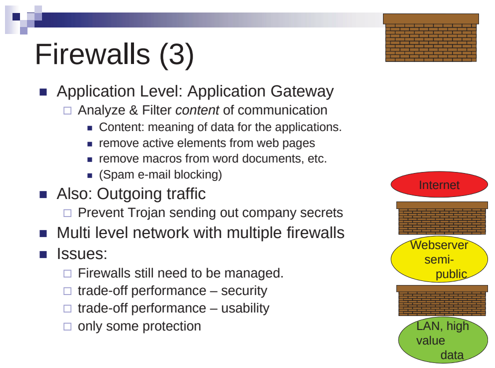
Firewalls We already briefly mentioned firewalls when taking about IP spoofing. As we have seen above, the effect of different attacks and the ease with which they are employed depends on the capabilities of an attacker. A local attacker is able to do much more easily cause more damage then a remote attacker. As such it makes sense to try to build barriers between different parts of the network. Firewalls do exactly this by filtering the traffic between two network, e.g. the internet and an organizations LAN, an organizations web services and its intranet, a single PC and the intranet, etc.



Firewalls (2)

- Network layer: Packet filtering
  - Do not let in packets for machine Y port X
    - If Y is not SSH server, block port 22
  - Do not let in packets with `local' source
    - block ip spoofing local addresses from outside
- Transport Level: e.g. Proxy Server
- Stateless vs Stateful firewall
  - E.g. Open TCP connections
  - Only allow response if query sent

20



Firewalls (3)

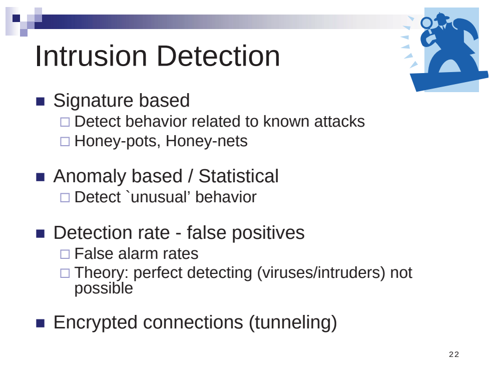
- Application Level: Application Gateway
  - Analyze & Filter *content* of communication
    - Content: meaning of data for the applications.
    - remove active elements from web pages
    - remove macros from word documents, etc.
    - (Spam e-mail blocking)
- Also: Outgoing traffic
  - Prevent Trojan sending out company secrets
- Multi level network with multiple firewalls
- Issues:
  - Firewalls still need to be managed.
  - trade-off performance – security
  - trade-off performance – usability
  - only some protection

Filtering can happen at different layers of abstraction (and thus at different network layers). For example a basic packet filtering firewall (working at the network layer) can help against IP spoofing of local addresses by outside attackers and can block access to ports (and services) that should not be accessible. If working at the TCP level, it will likely need to remember which sessions are open to be able to distinguish real responses from spoofed messages; it needs to be stateful. This can have a significant impact on the resources needed and thus the performance of the firewall. Going up to the application level one can try to block dangerous data or known threats; e.g. remove active components from web pages, macros from word documents, block downloaded files containing viruses, tag spam and phishing emails, etc. Of course this greatly increases the complexity of the firewall; instead of looking at single packages one needs to, understand the protocol being used, extract and reconstruct the data being sent, interpret and evaluate the data to determine whether it is harmful.

When using firewalls there is are clearly trade-offs being made; if we block traffic then availability and usability will go down. For example, by blocking port 22 of all machines except the public SSH server you help protect the network (e.g. a mis-configured machine vulnerable on port 22 would not be accessible from the outside) but also disallow other computers on the network from offering SSH connections. Also, the firewall will impact network performance with more advanced filtering requiring more effort thus adding to cost and/or leading to further loss of performance.

A good policy is thus needed to make firewalls useful; one that implements optimal trade-offs between protecting against network risks and keeping network services available. The risks, ways to detect attacks and network services needed change dynamically, making maintaining an effective firewall challenging. Note that even an optimal trade-off will not be perfect; it is not always possible to distinguish between legitimate traffic and traffic that is part of an attack. For example, the firewall can only inspect the traffic that it can see. If data is encrypted, a best practice for securing a connection, this limits the possibilities for the firewall. Summarizing, a firewall is a very useful tool but by itself will not be sufficient to protect a network.

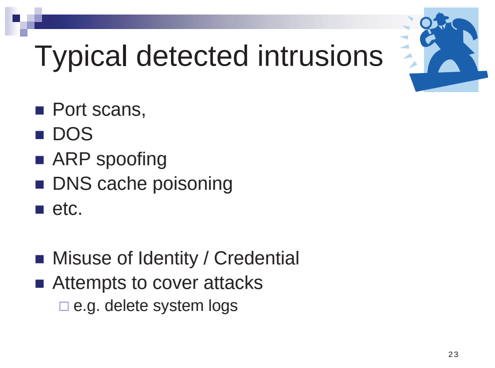
So how do we know what the attacks are and what type of traffic should be blocked? One way is to use Honey-pots and Honey-nets, interesting looking fake systems or networks, that collect data on attacks. Attackers are lured to these systems and monitored to reveal what vulnerabilities in which services they try to exploit.



Intrusion Detection

- Signature based
  - Detect behavior related to known attacks
  - Honey-pots, Honey-nets
- Anomaly based / Statistical
  - Detect `unusual' behavior
- Detection rate - false positives
  - False alarm rates
  - Theory: perfect detecting (viruses/intruders) not possible
- Encrypted connections (tunneling)

22



Typical detected intrusions

- Port scans,
- DOS
- ARP spoofing
- DNS cache poisoning
- etc.

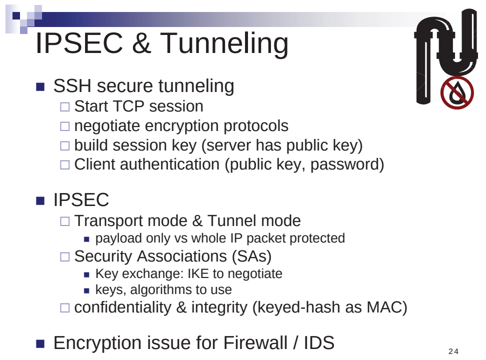
- Misuse of Identity / Credential
- Attempts to cover attacks
  - e.g. delete system logs

23

IDS Firewall are not perfect. So how can we find attackers that have found their way around our firewall (or are on our local network to begin with)? Intrusion detection systems (IDS) monitor the network to find (and possibly block) malicious activities. Such systems can be classified in two main classes. Signature based systems check for patterns in the traffic that match known attacks. Anomaly based systems look for 'abnormal' behaviour; anything that is not the usual behaviour on the network could indicate an attack. Like with firewalls, the intrusion detection can do basic analysis at the packet level or can reconstruct the meaning of the traffic to detect attacks at a higher level.

Not all attacks can be detected and not all detected anomalies (or signature matches) are attacks. A way to measure the quality of an IDS is to look at its detection rate versus its false positive (false alarm) rate. A perfect detection rate with no false positives is impossible. (Detecting whether a program stops is already not possible in general, let alone detecting malicious intent.) Again we are left with a trade-off between detection rate and the amount of (false) alarms that are raised and need to dealt with by the security officer (possibly with real attacks getting lost in the fray). Like firewalls, IDSs have become default technology for any organizations with a significant network.

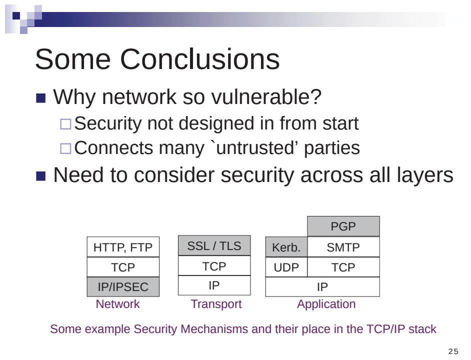
IPsec is a set of protocols for authentication and encryption of IP packets. It also supports mutual authentication of the agents involved. It works in a lower network layer than e.g. SSL and hence is transparent to the network applications. Transport mode: to protect confidentiality, the content of a packet is encrypted. The header is not modified (as not to influence the routing practises on the network). Integrity of the content and parts of the header of the packet can also be protected. (though this causes issues with Network Address Translation; a commonly used process for changing address on one network (e.g. all machines on your home LAN) to address(es) on another network (e.g. the single IP assigned to you by your internet provider). Tunnel mode: the entire packet, including header, is encrypted and then the result is send as content of a new IP packet.



IPSEC & Tunneling

- SSH secure tunneling
  - Start TCP session
  - negotiate encryption protocols
  - build session key (server has public key)
  - Client authentication (public key, password)
- IPSEC
  - Transport mode & Tunnel mode
    - payload only vs whole IP packet protected
  - Security Associations (SAs)
    - Key exchange: IKE to negotiate
    - keys, algorithms to use
  - confidentiality & integrity (keyed-hash as MAC)
- Encryption issue for Firewall / IDS

24



Some Conclusions

- Why network so vulnerable?
  - Security not designed in from start
  - Connects many `untrusted' parties
- Need to consider security across all layers

HTTP, FTP
TCP
IP/IPSEC
Network
SSL / TLS
TCP
IP
Transport
PGP
Kerb.
SMTP
UDP
TCP
IP
Application

Some example Security Mechanisms and their place in the TCP/IP stack

25

IPsec was developed as part of the IPv6 standard (though it can also be employed on IPv4). Still not all implementations of IPv6 include IPsec. Security associations are basically keys used to communicate, along with algorithms, protocols and settings used. IPSEC uses Internet Key Exchange (IKE) to set up security associations. It uses the DH key exchange we saw before to set up a shared session keys (or to be more precise a shared secret from which keys are derived). It can also use certificates (see later lecture) to authenticate parties.

When you initially setup an IPsec connection, you have some confidence that you will security communicate with the party that you setup the connection with; the confidentiality and integrity of the communication content can be protected. However, an authentication phase is still needed in which you ensure that the party you setup the connection with is actually a party you want to communicate. This introduces issues of policies, managing secrets(e.g. keys), etc. which we will see returning in following lectures (see e.g. the topics certificates& trust, authorization and access control.)

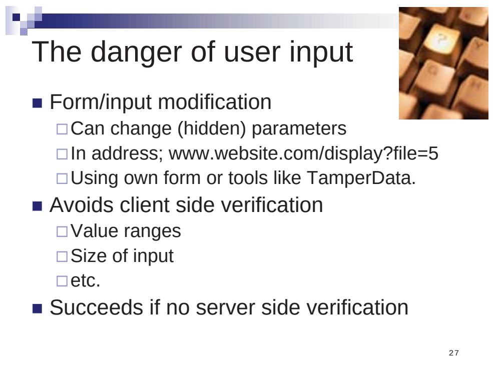
3.2 Web service security

Above we focussed on the (mechanisms of the) communication itself here we will look at the content of the communication, mostly from the server side as a main threat to the security of web services is part of this content; the input from the user.

Web services need to collect information from the user to do their job; the news article to display, the address to which to dispatch an order etc. Information collected through setting in links or explicitly entered in forms can easily be altered by a user(attacker). Sometimes changing the user-id in the address bar is already enough to get at the content of another user. Forms used to gather user input may contain checks (e.g. is an email address format correct, does the text entered in the telephone field contain only numbers, etc.). However, on the server end one should not rely on such checks as a user(attacker) can simply send its own (unchecked) data without using the form at all.

SQL injection The information used by the website is typically stored in an (SQL) database. to supply the correct information to a user, the web application will construct a query to this database using input from the user. For example, an application may look up the record for a given username-password combination resulting in the script line:

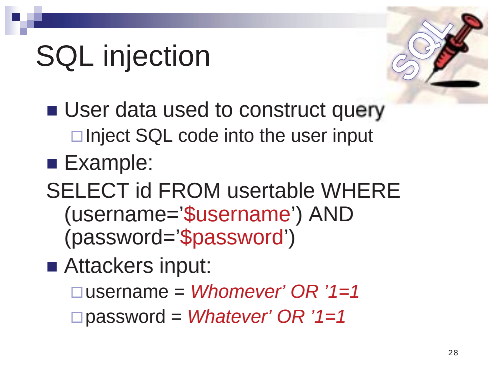
SELECT id FROM usertable WHERE
(username='$username') AND (password='$password')



The danger of user input

- Form/input modification
  - Can change (hidden) parameters
  - In address; www.website.com/display?file=5
  - Using own form or tools like TamperData.
- Avoids client side verification
  - Value ranges
  - Size of input
  - etc.
- Succeeds if no server side verification

27



SQL injection

- User data used to construct query
  - Inject SQL code into the user input
- Example:

SELECT id FROM usertable WHERE
(username='$username') AND
(password='$password')

- Attackers input:
  - username = *Whomever' OR '1=1*
  - password = *Whatever' OR '1=1*

28

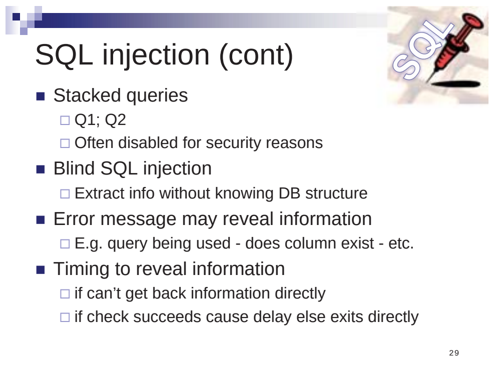
When Alice enters her username and password the query

SELECT id FROM usertable WHERE
(username='Alice') AND (password='1234')

is send to the database and the result is displayed to Alice. If, however, an attacker enters strange and unexpected data, the meaning of the constructed query can change - for example:

SELECT id FROM usertable WHERE
(username='Whomever' OR '1=1') AND (password='Whatever' OR '1=1')

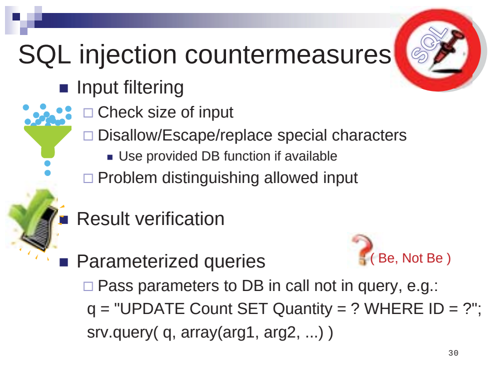
actually retrieves the entire usertable to show to the user. This is a basic example of an SQL injection attack; via the user input additional SQL code is injected into the query.



SQL injection (cont)

- Stacked queries
  - Q1; Q2
  - Often disabled for security reasons
- Blind SQL injection
  - Extract info without knowing DB structure
- Error message may reveal information
  - E.g. query being used - does column exist - etc.
- Timing to reveal information
  - if can't get back information directly
  - if check succeeds cause delay else exits directly

29



SQL injection countermeasures

- Input filtering
  - Check size of input
  - Disallow/Escape/replace special characters
    - Use provided DB function if available
  - Problem distinguishing allowed input
- Result verification
- Parameterized queries (Be, Not Be)
  - Pass parameters to DB in call not in query, e.g.:

```
q = "UPDATE Count SET Quantity = ? WHERE ID = ?";
srv.query( q, array(arg1, arg2, ...) )
```

30

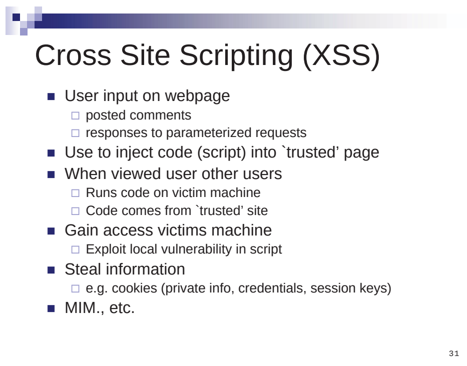
In some cases exploiting an SQL injection is very easy. For example with stacked queries, an SQL feature that is often disabled for security reasons, an injection can execute arbitrary commands on the data base not only change the current query. When doing a blind injection; where the structure of the data base and/or the queries constructed by the web application are not known some effort is first required to find this structure. Some web applications, when getting an error from the database, simply display the error to the user. This may be useful to the application developer for debugging purposes but will also provide a lot of valuable information to the attacker. For example, the query used, tables that exists/do not exist, etc. Sometimes the application filters the error only showing that some error occurred. Even only being able to see whether an error occurs, or even only how long it takes before the web application responds gives a way to extract information about/from the database. (See e.g. [**?**] for a local real world example.)

So what can be done about SQL injections? The web application can employ several countermeasures. The main one is input filtering. A basic step could be to check the size of the input; e.g. if the input is a pin it should only be a few characters long. This would already make it more difficult to construct successful SQL exploits. A stronger defense would be to ensure the user input is not interpreted as being part of the query by checking the input from the user for special characters (e.g. ') or sequences. The database API may actually have a filtering function that achieves this which should then be used so the web developer does not have to recode it (with a risk of errors and/or oversights).

When supported by the database, a way to safely get the input to the database is to use parameterized queries. Instead of building a query string which includes the users input, the query contains parameters and the user input is passed to the database as values to be used for the queries. In this way no confusion is possible between what is part of the query and what is user input. Thus the user input cannot change the query/inject code into it.

In addition to the input one could validate the output of the query. Similar to input size checking we can check whether the output matches our expectation; if we are expecting a single record and get back a whole table something is likely wrong and the results should not be used.

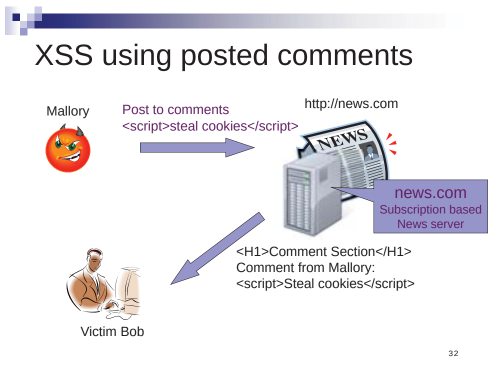
XSS Cross Site Scripting (XSS) is another example where user input form a danger to web applications and their users. In an XSS attack an attacker gets a user('s browser) to run code that seems to come from another (trusted) website. User input to the website is used to to inject code (a script) into a 'trusted' page. Coming from a trusted page, a victims machine will allow execution of the code which can then e.g. control communication with the trusted site (become a 'man in the middle'), steal information only meant for the trusted site (private/confidential information, credentials, sessions keys, etc.) or exploit local vulnerabilities to gain even more access to the victims machine.



Cross Site Scripting (XSS)

- User input on webpage
  - posted comments
  - responses to parameterized requests
- Use to inject code (script) into `trusted' page
- When viewed user other users
  - Runs code on victim machine
  - Code comes from `trusted' site
- Gain access victims machine
  - Exploit local vulnerability in script
- Steal information
  - e.g. cookies (private info, credentials, session keys)
- MIM., etc.

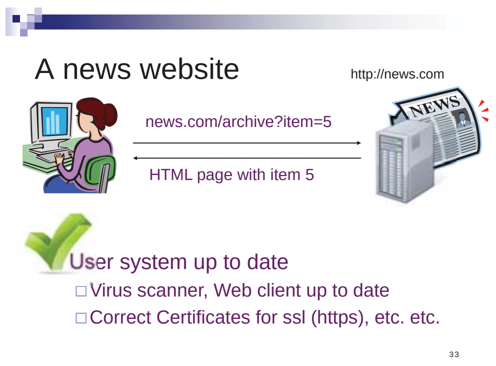
31



XSS using posted comments

32

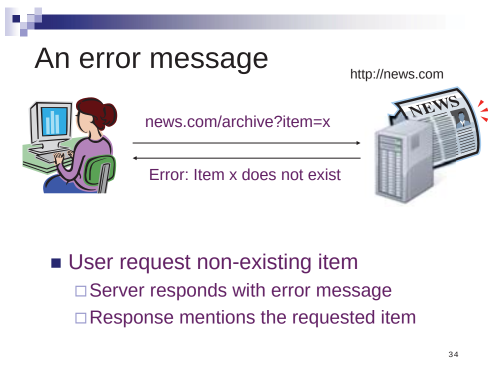
Consider a subscription new website 'news.com' where users can place comments with an article which are then shown to all users reading the article. An attacker Mallory could post a comment containing a script which would then be accepted by user Bob's browser as part of the 'news.com' website and thus has all the rights any script from news.com would have. It can read the cookies that news.com has set when Bob logged in and send them to Mallory. With those cookies Mallory can impersonate Bob at news.com.



A news website

User system up to date
- Virus scanner, Web client up to date
- Correct Certificates for ssl (https), etc. etc.

33

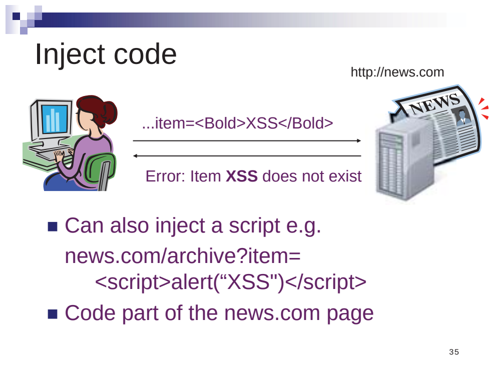


An error message

- User request non-existing item
  - Server responds with error message
  - Response mentions the requested item

34

A comment section is an obvious user input and news.com may, as with SQL injections employ input filtering, disallowing scripts (see also countermeasures below). However, there are also other, less obvious, user inputs that can be employed to launch an XSS attack. To indicate the selected article, news.com uses the ‘item’ parameter which should be set to the id number of some item. If a non-existent id number is used the website will report that the given number does not exist.

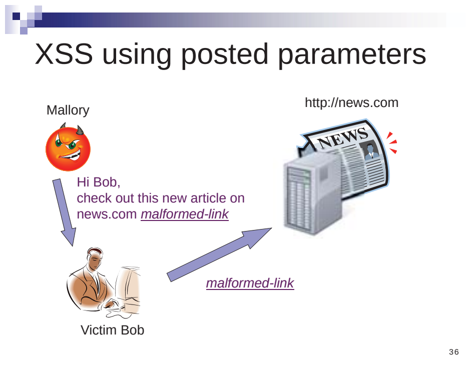


Inject code

http://news.com
...item=<Bold>XSS</Bold>
Error: Item **XSS** does not exist

- Can also inject a script e.g. news.com/archive?item=
  <script>alert(“XSS")</script>
- Code part of the news.com page

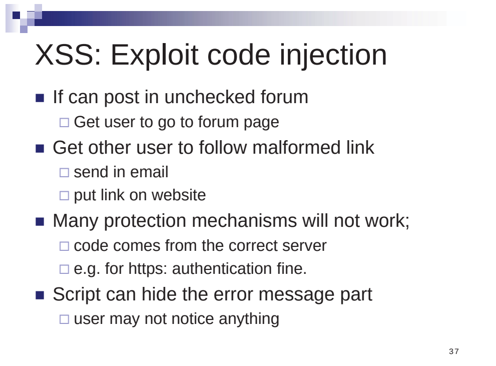
35



XSS using posted parameters

36

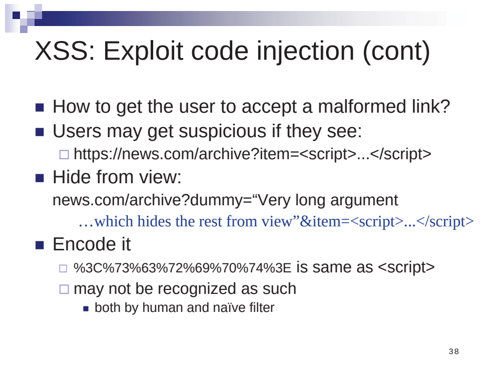
The error message is a way for a user to inject code; by setting item to some code instead of an article id that code will be injected into a news.com page showing the error message. The script can be used to hide the error message part so the page looks like a normal page. Now Mallory can now run code that seems to come from news.com but it runs on her own machine under her account. To get the code to run on Bob’s machine she needs to get Bob to follow the link she has constructed. To make sure Bob does not get suspicious she will likely need to hide the fact that the link contains a script.



XSS: Exploit code injection

- If can post in unchecked forum
  - Get user to go to forum page
- Get other user to follow malformed link
  - send in email
  - put link on website
- Many protection mechanisms will not work;
  - code comes from the correct server
  - e.g. for https: authentication fine.
- Script can hide the error message part
  - user may not notice anything

37

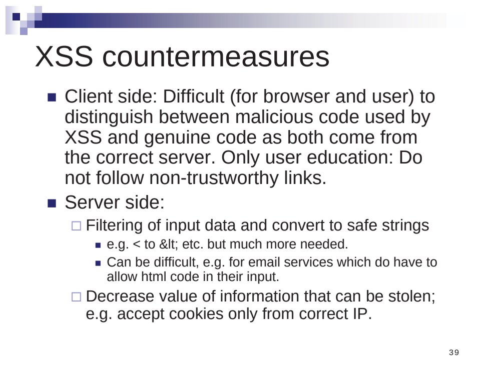


XSS: Exploit code injection (cont)

- How to get the user to accept a malformed link?
- Users may get suspicious if they see:
  - https://news.com/archive?item=<script>...</script>
- Hide from view:
  news.com/archive?dummy=“Very long argument
  …which hides the rest from view”&item=<script>...</script>
- Encode it
  - %3C%73%63%72%69%70%74%3E is same as <script>
  - may not be recognized as such
    - both by human and naïve filter

38

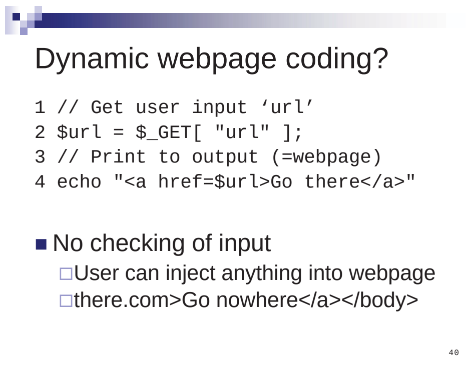
It is difficult to protect against XSS attacks on the user end; the code comes from the right server. One can educate the user not to follow untrusted links; even if they seem to go to ‘trusted’ sites.



XSS countermeasures

- Client side: Difficult (for browser and user) to distinguish between malicious code used by XSS and genuine code as both come from the correct server. Only user education: Do not follow non-trustworthy links.
- Server side:
  - Filtering of input data and convert to safe strings
    - e.g. < to &lt; etc. but much more needed.
    - Can be difficult, e.g. for email services which do have to allow html code in their input.
  - Decrease value of information that can be stolen; e.g. accept cookies only from correct IP.

39



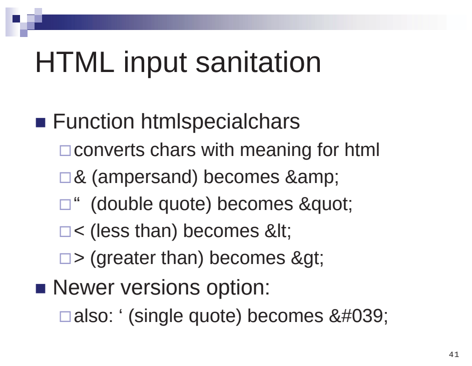
Dynamic webpage coding?

```
// Get user input ‘url’
$url = $_GET[ "url" ];
// Print to output (=webpage)
echo "<a href=$url>Go there</a>"
```

- No checking of input
  - User can inject anything into webpage
  - there.com>Go nowhere</a></body>

40

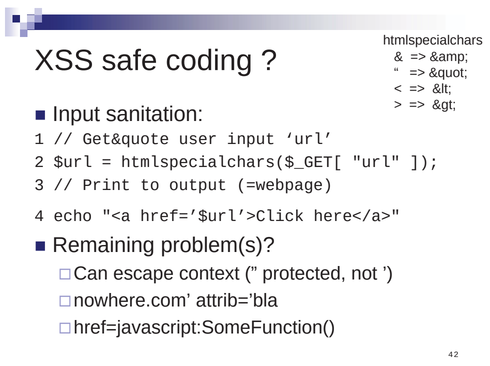
On the server side, as with SQL, input filtering is an important countermeasure. User input with special meaning should be translated into a ‘safe’ format. When coding a web application one should thus be very careful with user input. Note that filtering requires being able to distinguish between valid input and harmful content. This may be difficult, for example for a webmail application where the input is an email written in html. Removing dangerous parts but still allowing the user to give all the types of input can thus be very difficult. Web scripting languages typically provide some functions, e.g. the 'htmlspecialchars' function that replaces characters by their html encoding. This, however, is not sufficient to make the user input 'safe' in the example. There are tools which try to detect XSS vulnerabilities. Such tools aim to ensure functions such as 'htmlspecialchars' are applied in a sufficient way to make the input safe. As automatically checking user input usage against what is dangerous in a given settings is not easy such tools often miss vulnerabilities and/or create many false positive making them less convenient to use.



HTML input sanitation

- Function htmlspecialchars
  - converts chars with meaning for html
  - & (ampersand) becomes &amp;
  - " (double quote) becomes &quot;
  - < (less than) becomes &lt;
  - > (greater than) becomes &gt;
- Newer versions option:
  - also: ' (single quote) becomes &#039;

41



XSS safe coding ?

htmlspecialchars
& => &amp;
" => &quot;
< => &lt;
> => &gt;

- Input sanitation:

```
// Get&quote user input 'url'
$url = htmlspecialchars($_GET[ "url" ]);
// Print to output (=webpage)
echo "<a href='$url'>Click here</a>"
```

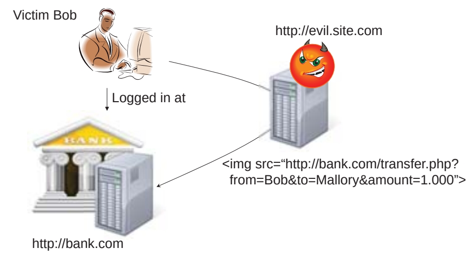
- Remaining problem(s)?
  - Can escape context (" protected, not ')
  - nowhere.com' attrib='bla
  - href=javascript:SomeFunction()

42

A general defense is to decrease the value of what can be stolen; e.g. accepting authentication cookies only from the right IP would make abuse of stolen cookies harder.

XSRF Cross-Site request forgery (XSRF) is another attack where user input (in this case requests) is misused. Where an XSS attack tricks a user('s browser) into trusting and running code, an XSRF attacks aims to send authorized but unintended requests to a (trusted) website. Consider a user logged in at bank.com but now visiting evil.site.com. On the evil site there is a link to an 'image'. The user's browser will follow this link to load the image. However, the link is not really an image but a request to bank.com to transfer money. As Bob is already logged in at bank.com the request will succeed.

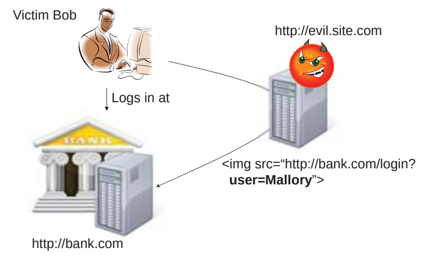
XSRF (Cross-Site request forgery)



```
<img src="http://bank.com/transfer.php?
from=Bob&to=Mallory&amount=1.000">
```

43

XSRF Login attack



```
<img src="http://bank.com/login?
user=Mallory">
```

44

A variation of the XSRF attack above has the same setting but now instead of sending a request as being Bob, Bob's session is actually replaced and he is now logged on as Mallory at bank.com. This type of attack may make sense if Mallory is trying to steal e.g. documents that Bob may upload to bank.com as these will now go to Mallory's account.

As a general conclusion for this section we can state: data is dangerous. Not only unknown programs or devices but also any data from an untrusted source can be a source of attacks.