**Ներածություն**

**(slide 2)** Http – hypertext transfer protocol, իրենից ներկայացնում է request/response մոդելի կոմունիկացիոն պրոտոկոլ որը հիմնված է client/server իրար հետ փոխազդեցության մեխանիզմի վրա։ Բրաուզերը գործում է որպես web կամ http կլիենտ։

Սերվերը իրենից ներկայացնում է որևէ ռեսուրս որը ունի մեքենայական մաս և դրա վրա աշխատող ծրագրային մաս (web server application): Web server application – ը ընդունում է request- ը և կլիենտին ուղարկում է HTTP response: Կլիենտը դա միջնորդ է սերվերի և user – ի միջև։ Որպես կլիենտ դիտարկում ենք բրաուզերը։ Կլիենտը HTTP request ուղարկելու նպատակով կապ է հաստատում սերվերի հետ, իսկ Http սպեցիֆիկացիան սահմանում է կանոններ թե ինչպես են web client – ի request – ները ստեղծվելու և ուղարկվելու սերվերին և ինչպես է սերվերը պատասխանելու դրանց։

Կլիենտը ուղարկում է request սերվերին` request – ի տիպի և url – ի տեսքով։ Request – ը իր մեջ պարունակում է նաև MIME որը պարունակում է կլիենտ, փոխացնվող տվյալների մասին ինֆորմացիա, նաև պրոտոկոլի վերսիայի մասին։

Սերվերը ուղարկում է response դրա մեջ ներառելով տվյալները, ստատուս կոդը, նաև response – ի տվյալների մասին մետաինֆորմացիա։ **(slide 2)**

**(slide 3)** TCP/IP – ի հիման վրա որը օգտագործվում է տվյալների փոխանակման համար (html էջեր, ֆայլեր, request – ների մշակման այլ արդյունքեր)։

HTTP – ն ունի հետևյալ հիմնական բնութագրերը

* Http – ն չի պահանջում մշտական միացման առայություն։ Յուրաքանչյուր նոր Http request – ի ժամանակ web client – ը բացում է նոր միացում, իսկ response – ը ստանալուց հետո փակում է միացումը։ Այսինքն կլիենտը և սերվերը փոխազդում են իրար հետ միայն request – ի և response – ի ժամանակ։
* Http – ն կախված չէ միջավայրից, այսինքն Http պրոտոկոլով կարելի է ուղարկել և ստանալ ցանկացած տիպի տվյալ, եթե web client – ը և սերվերը կարող են մշակել այդ տվյալները։ Կլիենտը և սերվերը պետք է նշեն կոնտենտի տիպը (MIME տիպը` Multipurpose Internet Mail Extensions, հանդիսանում է ստանդարտ որը բնութագրում է տվյալի տիպը։ Ունի ստանդարտների սպեցիֆիկացիա)։
* Http – ն stateless է, այսինքն սերվերը և կլիենտը փոխազդում են իրար հետ միայն request – ի և response – ի ժամանակ և իրար մասին, այդ թվում նաև request – ների մասին ինֆորմացիա չեն պահում։ **(slide 3)**

**(slide 4) Connection**

TCP/IP

HTTP – ն կիրառական մակարդակի պրոտոկոլ է՝ այսինքն ցանցային կապի մանր դետալները իր համար կարևոր չեն։ Կապի դետալները HTTP – ն թողնում է ցանցային պրոտոկոլներին։

Ինտերնետի միջոցով կապի հաստատումը հիմնված է ցանցային TCP/IP պրոտոկոլների ստեկի վրա։

Ցանցային պրոտոկոլներով տվյալների փոխանցումը շատ նման է սովորական փոստային առաքման մոդելին։

Ցանկացած համակարգիչ ինտերնետ ցանցի շրջանակում ունի ունիկալ IP(Internet protocol) հասցե: Բայց քանի որ տվյալներով փոխանակվում են ոչ թե համակարգիչները այլ դրանց վրա աշխատող application – ները մեզ ip – ից բացի պետք է նաև application – ի հասցեն/համարը կամ port - ը: Օր web service – ը լսում է 80 port – ը։ ip – ն և port – ը միասին կոչվում են սոկետ:

Ցնացի տեսանկյունից ամենավերևի մակարդակում աշխատում է HTTP – ն դրա տակից տեղափոխման պրոտոկոլ TCP – ն (transmision control protocol այս մակարդակում ավելանում է պորտը) դրա տակից ցանցային պրոտոկոլ IP – ն(համակարգչից համակարգիչ առանց պորտը նշելու)։ Ցանցային պրոտոկոլների միջոցով HTTP – ն փոխանցում է տվյալներ։ Http -ն կախված չէ ցանցային պրոտոկոլներից այն օգտագործում է ցանցային պոտոկոլները։

TCP – ից բացի կա նաև UDP (user datagrame protocol) և թե որն է պետք օգտագործել կախված է փոխանցվող տվյալների առանձնահատկություններից **(slide 4)**

**(slide 5)** Ռեսուրսի նույնականացման և դրա հետ կապ հաստատելու համար HTTP- ն օգտագործում է URI (Uniform Resource Identifier) ։

URI = "http:" "//" host [ ":" port ] [ abs\_path [ "?" query ]]

Uri – ն իր մեջ ներառում է URL(uniform resource location) և URN - uniform resource name)

Սերվերի IP հասցեն ստանալու համար օգտագործվում է URL – ը(հոստը), որը նույն IP - ի թվային արտահայտությունն է կոնվերտացված տքեստի։ Հոստերի անունները կոնվերտացվում են IP հասցեների Domain Name Service կամ կրճատ DNS – ի միջոցով։ Իսկ 80 port – ը uri – ում կարող է նշվել, իսկ չնշվելու դեպքում default կդրվի 80 արժեքը, քանի որ http – ի ժամանակ վեբ սերվիսը լսում է 80 port – ը։

**(slide 5)**

**(slide 6)**

DNS – ի աշխատանքը ունի հետևյալ սխեմատիկ տեսքը։ բրաուզերում user-ը գրում է ռեսուրսի հասցեն։ DNS – ը url – ը համադրոմ է ip հասցեի հետ և տվյալ ip – ով կապ է հաստատվում սերվերի հետ:

**(slide 6)**

**(slide 7) Web միջնորդներ, proxy, gateway (շլյուզ)**

Կլիենտի և սերվերի միջև կարող են աշխատել միջնորդ ռեսուրսներ պրոքսի սերվերի, կամ շլյուզի տեսքով, առանց պրոքսիի կլիենտը և սերվերը անմիջականորեն աշխատում են իրար հետ, պրոքսիի դեպքում դրա միջոցով։ Կարելի է ասել որ միջնորդ ռեսուրսը միաժամանակ և կլիենտ է և սերվեր քանի որ այն միևնույն ժամանակ ընդունում է request – ներ և ուղարկում response – ներ։

Պրոքսիները իրար են կապում նույն պրոտոկոլով աշխատող տարբեր application – ներ, իսկ շլյուզները տարբեր պրոտոկոլներով աշխատող application – ներ։ Շլյուզը աշխատում է որպես պրոտոկոլների ինտերպրետատոր։

Բայց պրակտիկայում շատ հաճախ պրոքսիների և շլյուզների միջև տարբերությունը հստակ չէ քանի որ շատ հաճախ կլիենտը և սերվերը օգտագործում են HTTP – ի տարբեր վերսիաներ և պրոքսին ստիպված է լինում համադրել տարբեր վերսիաները և կատարել շլյուզներին բնորոշ ինտերպրետացիոն գործողություններ։

Միջնորդ ռեսուրսները օգտագործվում են անվտանգության, արդյունավետության բարձրացման, տարբեր սերվիսներ ավելացնելու օր՝ կոնտենտի հասանելիության համար ֆիլտրացիայի գործողություններ, քեշավորում, routing **(slide 7)**

**(slide 8)** **HTTP messages**

Http client Http server փոխազդեցությունը՝ տվյալների փոխանցումը, տեղի է ունենում HTTP – ում պարունակվող տվյալների բլոկներով որոնք կոչվում են Http message – ներ։

Http message – ը իրենից ներկայացնում է սիմվոլային հաջորդականություն այսինքն տեքստ: Կան 2 տիպի message – ներ request message և response message

HTTP message – ների միջոցով տվյալներ են փոխանցվում կլիենտի և սերվերի միջև։ HTTP message – ները և request–ի և response-ի դեպքերում փոխանցվում են մեկ ընդհանուր ֆորմատով (RFC 822), որը իր մեջ պարունակում է 4 բաղադրիչ:

* Message – ի սկզբնական տողը որը բնութագրում է request – ը կամ response – ի ստատուսը` այսինքն թե ինչ է պետք անել request – ի հետ, կամ ինչ է տեղի ունեցել response – ի հետ։
* Header fields որոնք կարող են լինել 0 և ավելի քանակով և իրենցից ներկայացնում են name value առանձնացված ։ - ով: Header – ներով ավելացվում են լրացուցիչ տվյալներ request – ին կամ response – ին։ Պարրամետրեր են որոնք բնութագրում են request – ը կամ response - ը
* Դատարկ տող որով նշվում է header – ների ավարտը
* Body (optional է) HTTP – ի օգտակար բեռն է, կարող է պարունակել ցանկացած տվյալ, request – ների body – ն փոխանցում է տվյալ սերվերին իսկ սերվերը body – ով կլիենտին է հետ վերադարձնում տվյալներ։ Ի տարբերություն սկզբնական տողի և headr – ների որոնք տեքստեր են body – ն տեքստից բացի կարող է պարունակել նաև բինար data(նկար, վիդեո, ծրագրային app - ներ և այլն) **(slide 8)**

**(slide 9)**

Messag – ները ունեն հետևյալ տեսը

**(slide 9)**

**(slide 10) Request messages**

HTTP request – ը դա message – ներ են որոնք կլիենտը ուղարկում է սերվերին։ Start line - ը Իր մեջ պարտադիր պետք է պարունակի մեթոդի անունը, ռեսուրսի հասցեն, (URI – ը կամ միայն URN - ն) HTTP version – ը։

Start line (request line) – Method Request-URI HTTP-Version

Մեթոդի անունը պետք է լինի մեծատառերով։

Uri – ն կարող է լինել հետևյալ ձևերի՝

«\*» գտագործվում է երբ request – ը ուղղված է ոչ թե կոնկրետ ռեսուրսի այլ հենց սերվերին։ Թույլատրված է եթե մեթոդը ռեսուրսին չի վերաբերում օր Options \* HTTP/1.1

absoluteURI երբ ուղարկվում է ամբողջ URI – ն հոստի հետ միասին

abs\_path հասցեն առանց հոստի

**HTTP version**

Http – ի վերսիայի նումերացիան կատարվում է <major><minor> սխեմայով

0.9 – էքսպերիմենտալ տարբերակ

1, 1.1 վերսիաներում կատարվել են օպտիմիզացիաներ, սեմանտիկ բար ելավումներ: Լայն տարածում ունի 1․1 բայց կա արդեն 2․0 և 3․0 – ի պրոտոտիպը։ **(slide 10)**

**(slide 11) HTTP methods**

HTTP/1.1 – ում կարող են ավելացվել ցանկացած քանակությամբ մեթոդներ, բայց ընդհանուր ստանդարտներին հետևելու համար օգտագործվում են այն մեթոդները որոնց կիրառման նպատակը նկարագրված է պրոտոկոլի սպեցիֆիկացիայում։

Հարկ է նշեն որ բոլոր մեթոդների անունները զգայուն են ռեգիստրին և պետք է լինեն միայն մեծատառերով։

Առանձնացնում են այսպես կոչված safe մեթոդներ GET HEAD OPTIONS որոնց արդյունքում սերվերի վրա ոչինչ չի փոխվում։

GET – օգտագործվում է նշված URL - ով սերվերից տվյալներ ստանալու համար։ Պարամետրերը փոխանցվում են URL – ի միջոցով GET request – ները պետք է միայն տվյալներ ստանան և որևէ կերպ չպետք է ազդեն տվյալների վրա։

HEAD – մեթոդը աշխատում է GET – ի նման այն տարբերությամբ որ վերադարձնում է միայն status line – ը և header- ները, առանց body - ի։ Կարող է օգտագործվել ռեսուրսի մասին

PUT – օգտագործվում է սերվերի վրա տվյալներ ավելացնելու / գրելու համար։ Սերվերը վերցնում է request – ի body – ն և օգտագործում է այն նոր data ստեծելու կամ արդեն գոյություն ուեցող data - ն փոխելու համար։

POST – օգտագործվում է տվյալները սերվերին ուղարկելու ժամանակ, օրինակ վերցված HTML ֆորմաներից։

TRACE – օգտագործվում է HTTP request – ի պարունակությունը տեսնելու համար։ HTTP request – ը մինչև սերվերին հասնելը կարող է անցնել պրոքսի սերվերներով, շլյուզներով և այլ։ Դրանցից յուրաքանչյուրը կարող է փոխել HTTP request – ը։ TRACE մեթոդը կլիենտին թույլ է տալիս տեսնել request – ի վերջնական տեքը, երբ այն հասնում է սերվերին։

OPTIONS – օգտագործվում է սերվերի կողմից support արվող պարամետրերի մասին տվյալներ ստանալու համար։ Օրինակ option - ով կարելի է ստանալ թե սերվերը ինչ մեթոդներ է support անում։

DELETE – օգտագործվում է սերվերից որևէ տվյալ ջնջելու համար։

CONNECT – օգտագործվում է սերվերի հետ կապ հաստատելու համար **(slide 11)**

**(slide 12)** **Իդեմպոտենտություն**։ HTTP մեթոդը համարվում է իդեմպոտենտ երբ նույն request – ը որքան անգամ էլ կատարվի միշտ ունենում է միևնույն արդյունքը և չի փոխում սերվերի վիճակը։ Այսինքն իդեմպոտենտ մեթոդը չպետք է ունենա կողմնակի ազդեցություններ։ Այս տեսանկյունից իդեմպոտենտ են բոլոր մեթոդները բացի post – ից։ Իդեմպոտենտությունը կախված է չէ վերադարձվող ստատուս կոդից նույնիսկ եթե դրանք տարբեր են օր մեկ անգամ delete – ից հետո մյուս նույն delete –ի request ները կվերադարձնեն 404 բայց DELETE – ը համարվում է իդեմպոտենտ **(slide 12)**

**(slide 13) Respone messages**

Reques – ը message – ներ են որոնք սերվերը ուղարկում է կլիենտին։

Start line - ը Իր մեջ պարտադիր պետք է ներառի HTTP version – ը և ստատուս կոդը

Start line (Status-Line) = HTTP-Version Status-Code Reason-Phrase

**(slide 13)**

**(slide 14)** **HTTP Status Codes**

Սերվերի response – ում status code էլեմենտը իրենից ներկայացնում է եռանիշ թիվ որտեղ առաջին թիվը սահմանում է status – code – ի տեսակը։ Բուլոր ստատուս կոդերը ունեն նաև տեքստային նկարագրություն։

1 – ով սկսվող կոդերը ինֆորմատիվ ստատուս կոդերն են request – փոխանցման պրոցեսսի մասին։

2 – ով սկսվող կոդերը նշանակում են որ request – ը հաջողությամբ մշակվել է։

3- ով սկսվող կոդերը redirect – ի վերահասցեագրման մասին են

4- ով սկսվող կոդերը client error – ներն են

5- ով սկսվող կոդերը server error – ներն են **(slide 14)**

**(slide 15)** Հաճախ հանդիպող ստատուս կոդերը հետևյալներն են **(slide 15)**

**(slide 16)** **HTTP header fields**

Header field – երը նախատեսված են request – ին կամ response – ին լրացուցիչ տվյալներ ավելացնելու համար։ Պարրամետրեր են որոնք բնութագրում են request – ը կամ response - ը

Գոյութոյւն ունեն header – ների 4 տեսակներ

General-header – որոնք ունեն նույն կիրառությունը և request-ների և response – ների համար

Client request-header – որոնք կիրառելի են միայն request – ների համար

Server response-header - որոնք կիրառելի են միայն response – ների համար

Entity-header – որոնք սահմանում են entity – ի body – ի մասին մետաինֆորմացիա, իսկ եթե body – ն բացակայում է ապա request – ի մեջ նշված ռեսուրսի մասին։ **(slide 16)**

**(slide 17)** **General headers**

**Cache-Control** – օգտագործվում է քեշավորման istruction – ներ սահմանելու համար, որնց պետք է ենթարկվի ամբողջ քեշավորման մեխանիզմը։ և սերվերը և կլիենտը կարող են սահմանել cache-control header – ը և դրան տալ պարամետրեր։ քեշավորման դիրեկտիվները նշվում են հաջորդաբար և առանձնացվում են ստորակետներով։

**Pragma**

Քեշավորման header որը հնացել է

**Connection**

Կլիենտին և սերվերին թույլ է տալիս սահմանել կապի պարամետրեր

Connection: close – նշանակում է որ կապը կփակվի response – ը ստանալուց հետո

Connection: keep-alive – դեֆոլտով HTTP 1․1 – ն օգտագործում է մշտական կապի պահպանում, իսկ 1․0 – ի դեպքում պետք է նշել keep-alive header – ը

**Date**

Ցույց է տալիս թե երբ է ստեղծվել message – ը

Trailer

Թվարկում է header – ները որոնք կան message – ի մեջ

**Transfer-Encoding**

Ստացողին հայտնում է թե message – ի համար ինչ encoding է օգտագործվել, որպեսզի այն անվտանգ տեղափոխվի

**Upgrate**

Կլիենտը նշում է լրացուցիչ պրոտոկոլներ որոնք որ support է անում

**VIA**

Ցույց է տալիս թե ինչ միջնորդներով (պրոկսի սերվեր, շլյուզ) է անցել message – ը։

**Warning**

Փոխանցվում է լրացուցիչ տեղեկատվություն message – ի ստատուսի կամ դրա կոնվերտացիայի մասին, որը կարող է չերևալ message – ում։ Response – ը կարող է պարունակել 1-ից ավելի warning header: **(slide 17)**

**(slide 18) Client request headers**

Կլիենտի request – ի header – ները կարելի է բաժանել հետևյալ խմբերի՝

* Informational Headers
* Accept Headers
* Security Headers
* Conditional Headers
* Proxy-Authorization **(slide 18)**

**(slide 19) Request Informational Headers**

Սրանք այն header – ներն են որոնք կիրառվում են request – ի հետ։ Դրան ցույց են տալիս տվյալներ request – ի մասին։

Client-IP – ցույց է տալիս կլիենտի IP

Host – նշվում է սերվերի հոստը և պորտը, որին ուղղված է request – ը

Referer – նշվում է URI հղումը, որից վերցվում է URL հասցեն

From: email – պարունակում կլիենտի հետ կապված էլեկտրոնային հասցե, կարող է օգտագործվել անցանկալի աղբյուրներից ստացվող request – ները ֆիլտրելու համար

User-Agent – ցույց է տալիս թե ինչ application է կատարել request – ը **(slide 19)**

**(slide 20) Request Accept Headers**

Accept: type/subtype [q=value] - նշվում է մեդիայի տիպերը որոնք ընդունելի են response – ի համար և որակի ոչ պարտադիր արժեքը 0 – 1, եթե նշվում է արժեք ապա այն դառնում է երկրորդական նվազման նախապատվությամբ,

Accept-Charset: character\_set [q=value] – նշվում է թույլատրելի սիմվոլները

Accept-Encoding։ encoding types – սահմանափակում է թույլատրելի encoding – ը response – ի համար

Accept-Language: language [q=value] - նշվում է թույլատրելի լեզուները **(slide 20)**

**(slide 21) Request Security Headers**

Authorization – արժեքը պարունակում է ավտորիզացիայի տվյալներ

Cookie – իր մեջ պահում է name-value **(slide 21)**

**(slide 22) Request Conditional Headers**

Expect: expectation-extension – թվարկում է սերվերի սպասելի վարքագիծը, օր կարող է լինել նշված ստատուս կոդ իր message – ով 100-continue, ոթե սերվերը նշված արժեքը support չի անում ապա կվերադարձնի 417 error

If-Match: tag – ստանալ պատասխան եթե նշված թեգը համապատասխանում է տվյալի թեգին

If-Modified-Since: date չտալ պատասխան եթե url – ն չի փոխվել նշված ամսաթվից հետո

If-Unmodified-Since: date ստանալ պատասխան եթե url – ն չի փոխվել նշված ամսաթվից հետո

Range – սերվերից ստանալ տվայլներ դիապազոնով եթե նման ֆունկցիա սերվերը support է անում **(slide 22)**

**(slide 23) Proxy-Authorization**

Max-Forwards: n – օգտագործվում է TRACE, OPTIONS մեթոդների հետ, միջնորդների՝ պրոկսի սերվերների և շլյուզների քանակը սահմանափակելու համար, օգտագործվում է անվերջ ցիկլերից խուսափելու համար

Proxy-Authorization – թույլ է տալիս կլիենտն անցնել նույնականացում պրոքսի սերվերի վրա, որպես արժեք նշվում ավտորիզացիոն տվյալներ **(slide 23)**

**(slide 24) Server response headers**

Կլիենտի request – ի header – ները կարելի է բաժանել հետևյալ խմբերի՝

* Informational Headers
* Security Headers
* Negotiation headers

**Response Headers**

Սրանք այն header – ներն են որոնք կիրառվում են response – ի հետ։ Դրանց միջոցով կլիենտին հաղորդվում են լրացուցիչ տվյալներ response – ի մասին։

Age: seconds - Ցույց է տալիս response – ի ստեղծման պահից անցած ժամանակը

Retry-After – օգտագործվում է 503 response – ի հետ և ցույց է տալիս թե որքան ժամանակ սերվիսը կլինի անհասանելի

Server – ցույց է տալիս սերվերի ծրագրային ապահովման մասին տվյալներ

**Response Security Headers**

Proxy-Authenticate – 407 – ի մաս, որը պահանջում է պրոկսի սերվերի իսկության ստուգում

Set-cookie

WWW- Authenticate – ներառվում է 401 – ի մեջ, կլիենտին ներկայացվում է ավտորիզացիոն պահանջներ

**Response Negotiation Headers**

Http – ն ունի header – ներ որոնք թույլ են տալիս սերվերին տալ տվյալներ ռեսուրսի մասին այսպես ասած դիալոգ պահելու, պայմաններ սահմանելու համար։

Accept-Ranges – սերվերը ցժույց է տալիս որ ընդունում է դիապազոնի request – ներ

Vary – այլ header – ների հավաքածու որոնք դիտարկում է սերվերը և որոնք կարող են բերել response – ի փոփոխման, այսինքն header – ներ որոնք սերվերը կարող է ընտրել response – ի մեջ ներառման համար։ **(slide 24)**

**(slide 25) Entity headers**

Կլիենտի request – ի header – ները կարելի է բաժանել հետևյալ խմբերի՝

* Informational Headers
* Security Headers
* Negotiation headers **(slide 25)**

**(slide 26) Entity Headers**

Այս header – ները ցույց են տալիս response – ի body – ի մասին լայն ինֆորմացիա, այլ կերպ ասած ինֆորմացիա թե կլինտը ինչի հետ գործ ունի։

**Entity Information Headers**

Allow – թվարկում է մեթոդները որոնք կարող են կիրառվել տվյալ entity – ի համար։

Location կլիենտին հաղորդում է թե որտեղ է գտնվում entity – ն

**Entity Caching Headers**

Տալիս են տվյալներ քեշավորվող օբյեկտի մասին

Expires ցույց է տալիս ամսաթիվ որից հետո response – ը expire է լինում, գործողության ժամկետը անցնում է

Last-Modified – ամսաթիվ երբ entity – ն փոփոխվել է վերջին անգամ **(slide 26)**

**(slide 27) Entity Content Headers**

Այս header – ները ցույց են տալիս entity – ի պարունակության մասին տվյալներ

Content-Encoding – body – ի encoding – ի մասին տվյալ

Content-Language - body – ի լեզվի մասին տվյալներ

Content-Length - body – ի չափը

Content-Location – body – ի գտնվելու վայրը որը նշվում է URI – ով

Content-MD5 – body – ի digest-ը md5 կդավորմամբ, որը հաշվարկվում է body – ի պարունակությամբ

Content-Range: bytes – body – ի դիապազոնը

Content-Type: type/subtype – body – ի մեդիա տիպը **(slide 27)**

**(slide 28) Cache Control**

Օգտագործվում է այն application – ներում որոնց աշխատանքը կարող է օպտիմալանալ քեշավորման օգտագործման հաշվին։ Քեշավորումը ապահովելու մեխանիզմը համապատասխան դիրեկտիվների օգտագործումն է Cache-Control header – ի հետ, որոնք սահմանում են քեշավորման ալգորիթմներ։

**Cache-Control: General headers**

No-cache – քեշը չպետ է օգտագործի response - ը հաջորդ request – ի ժամանակ առանց սերվերի վրա ստումից հետո։

No-store – քեշը request – ի կամ response - ի մասին ոչինչ չպետք է պահի։

max-age = seconds կլիենտը պատրաստ է ընդունել response որի age – ը չի գերազանցում նշված արժեքը վայրկյաններով։

no-transform չի կոնվերտացնում entity – ի body – ն։

**Cache-Control: Request headers**

max-stale [=seconds] կլիենտը պատրաստ է ընդունել response – ը որը արդեն expired է եղել, բայց որի expired – ի ժամկետը չի անցնում նշված արժեքը վայրկյաններով։

min-fresh=seconds կլիենտը պատրաստ է ընդունել response որի կյանքի տևողությունը (expired) փոքր չէ նշված արժեքից վայրկյաններով։

only-if-cached նոր տվյալներ չի դուրս բերում, տվյալը ստացվում է միայն եթե այն քեշավորված է և նոր վերսիայի ստուգման համար ելակետային սերվերի հետ կապ հաստատելու կարիք նույնպես չկա։

**Cache-Control: Response headers**

Public - ցույց է տալիս որ response – ը կարող է քեշավորվել ցանկացած քեշի կողմից

Private - ցույց է տալիս որ response – ը կամ դրա մի մասը նախատեսված է միայն մեկ կլիենտի համար և չպետք է քեշավորվեն ընդհանուր քեշով

No-cache – քեշը չպետ է օգտագործի response - ը հաջորդ request – ի ժամանակ առանց սերվերի վրա ստումից հետո։

No-store – քեշը request – ի կամ response - ի մասին ոչինչ չպետք է պահի։

no-transform չի կոնվերտացնում entity – ի body – ն։

must-revalidate – քեշը պետք է ստուգի հնացած տվյալների ստատուսը դրանց օգտագործումից առաջ, իսկ expire – ից հետո տվյալը չպետք է օգտագործվի։

proxy-revalidate նույնն է ինչ որ must-revalidate – ը այն տարբերությամբ որ ․․․․․․․

max-age = seconds կլիենտը պատրաստ է ընդունել response որի age – ը չի գերազանցում նշված արժեքը վայրկյաններով։

s-maxage = seconds օգտագործվում է max-age – ով կամ expires – ով սահմանված maxage – ը override անելու համար **(slide 28)**

**(slide 29) URL Encoding**

Url – ն ուղարկվում է ASCII սիմվոլների օգտագործմամբ, բայց URL – ի մեջ կարող են լինել նաև սիմվոլներ որոնք չեն մտնում ASCII սիմվոլների ցանկի մեջ։ Url – ի կոդավորման համար այդ սիմվոլները փոխվում են %-ով որին հաջորդում 2 հատ 16 – ական թիվեր

Կա ցուցակ որի մեջ նշված են ASCII սիմվոլները և դրանց կոդավորված տարբերակները, որոնք կարելի է օգտագործել URL – ի մեջ։ **(slide 29)**

**(slide 30) HTTP Security**

HTTP security – ի նպատակը դա տվյալների անվտանգության մակարդակի բարձրացումն է։ HTTP – ում տվյալները փոխանցվում են բաց տեքստային տեսքով որը այդ տվյալները դարձնում է խոցելի։

**Personal Information Leakage**

Քանի որ կլիենտի տրամադրության տակ կան կարևոր անձնական տվյալներ, պետք է պահպանել որոշ կանոններ HTTP պրոտոկոլից դեպի այլ ռեսուրսներ տվյալների արտահոսքը կանխելու համար։

* Բոլոր կոնֆիդենցիալ տվյալները պետք է պահվեն սերվերի վրա կոդավորված տեսքով
* Սերվերի վրա աշխատող ծրագրային ապահովման մասին տվյալները հասանելի դարձնելը սերվերը ավելի խոցելի է դարձնում
* Պրոքսի սերվերները պետք է ձեռնարկեն հատուկ միջոցներ փոխանցվող request – ում` Form կամ Referer header – ով տվյալ չթողնելու համար From header – ով փոխանցված տվյալը կարող է հակասել user – ի տվյալների կոնֆիդենցիալության ապահովմանը, այսինքն այն կամ պետք է չփոխանցել եթե user – ը չի կարող այն անմիջականորեն փոփոխել

Կլիենտը չպետք է referer header – ը փոխանցի չպաշտպանված HTTP request – ով եթե հղման հասցեն ստացել է պաշտպանված պրոտոկոլով

* Ֆորմաներից ստացված տվյալները չպետք է սերվերին փոխանցվեն GET մեթոդվ քանի որ փոխանցված տվյալները հասանելի կլինեն URI – ում

**File and Path Names Based Attack**

Սերվերը պետք է թույլ չատ որ request-uri – ի կառուցվածքում լինեն այնպիսի տարրեր որոնք կարող են տարբեր օպերացիոն համակարգերում բերել անցանկալի հետևանքների։ Օր windows – ում path - ում ․․ - ի միջոցով հասանելիություն ենք ստանում ավելի բարձր մակարդակի կատալոգի

**DNS Spoofing**

HTTP օգտագործող կլիենտները օգտագործում են նաև DNS – ներ, որի արդյունքում DNS – ները դառնում են հարձակման թիրախ։ Հարձակման էությունն է IP հասցեների և դոմենային անունների սխալ համադրությունը։

**Location Headers and Spoofing**

Եթե սերվերը օգտագործում է 1 – ից ավելի organization – ներ ապա պետք է լրացուցիչ ստուգվի response – ի Location և content Location header – ները որոնք գեներացվում են այդ organizaion – ների հսկողությամբ, որպեսզի համոզվենք որ դրանք չեն փորձում ռեսուրսը ոչ վալիդ դարձնել։

**Authentication Credentials**

Կլիենտները ավտորիզացիոն տվյալները կարող են պահել անորորշ ժամանակահատվածով։ Http – ն սերվերին չի տալիս հնարավորություն նշել կլիենտին հրաժարվել ավտորիզացիոն քեշերից, որը անվտանգության տեսանկյունից խնդիր է։

**Proxies and Caching**

Պրոկսի սերվերները հանդիսանալով միջնորդներ հասանելիություն են ստանում կոնֆիդենցիալ տվյալների և դրանով դառնում են հարձակման թիրախ։ Քեշավորող պրոքսիները լրացուցիչ վտանգ են ներկայացնում, դրա համար քեշի պարունակությունը պետք է լրացուցիչ պաշտպանվի։ **(slide 30)**

**(slide 31) HTTPS**

HTTP – ի ամենաանվտանգ տարբերակը դա HTTPS – ն է որի ժամանակ reuest – ը և response – ը ուղարկվելուց առաջ կոդավորվում են SSL/TLS ծածկագրող պրոտոկոլի/մեխանիզմի միջոցով։

SSL (secure sockets layer) – սկզբից տեղի է ունենում ռեսուրսի նույնականացում սերտիֆիկատի միջոցով, որը ռեսուրսը ուղարկում է բրաուզերին, որից հետո տեղի է ունենում կոդավորված տվյալների փոխանցում։

TLS (transport layer security) – ssl – ի արդիականացված տարբերակն է, ներկա պահի վերսիան TLS 1.2, 1.3

HTTPS – ի դեպքում օգտագործվում է TCP 443 պորտը։ Նրկա պահին HTTPS – ը support է արվում համարյա թե բոլոր բրաուզերների կողմից։

Անվտանգությունը ապահովվում է SSl/TLS պրոտոկոլների միջոցով որոնք ունեն պաշտպանության 3 մակարդակ։

Հիմնված է այն բանի վրա որ, կլիենտի և սերվերի համար գեներացվում է ընդհանուր key, որի միջոցով տեղի է ունենում փոխանցվող տվյալների կոդավորումը։ Key – ը գեներացվում է յուրաքանչյուր սեանսի համար։ Սերվերը ունի թվային սերտիֆիկատ որի միջոցով նույնականացվում է սերվերը։ HTTPS – ով կապ հաստատելուց հետո բրաուզերը ստուգում է սերտիֆիկատի իսկությունը և դրանից հետո միայն շարունակվում է տվյալների փոխանակումը։ Օրինակ searching service – ները բարձրացնում են HTTPS պրոտոկոլով աշխատող կայքերի նախապատվությունը, իսկ բրաուզերները հատուկ զգուշացնում են եթե տվյալները փոխանցվելու են ոչ HTTP պրոտոկոլով **(slide 31)**