

Q1 *Suit of Armor Around the World*

(4 points)

You are tasked with securing The Avengers' internal network against potentially malicious protocols! For each type of firewall and set of traffic, state whether the firewall is able to achieve the desired functionality with perfect accuracy. **Assume that IP packets are never fragmented.** All connections that are not mentioned can be either allowed or denied.

If you answer Possible, briefly (in 3 sentences or less) how the firewall should operate to achieve the desired effect. If you answer False, provide a brief justification for why it isn't possible.

Q1.1 (1 point) **Desired Functionality:** Block all inbound TCP connections. Allow all outbound TCP connections.

Firewall: Stateless packet filter

☐ Possible

☐ Not Possible

Q1.2 (1 point) **Desired Functionality:** Allow all outbound TLS connections. Block all outbound TCP connections that aren't running TLS.

Firewall: Stateful packet filter

☐ Possible

☐ Not Possible



(Question 1 continued...)

Q1.3 (1 point) **Desired Functionality:** Allow outbound DNS requests. Block inbound DNS responses. Assume that name servers always listen on port 53.

Firewall: Stateless packet filter

☐ Possible

☐ Not Possible

Q1.4 (1 point) **Desired Functionality:** Block all HTTP traffic that contains the literal string **Ultron**. Allow all other HTTP traffic.

Firewall: TCP proxy

☐ Possible

☐ Not Possible

Q2 Access Control

(6 points)

EvanBot uses **MailBot**, a web app that helps summarize all the emails Evanbot gets in a day.

EvanBot needs to delegate access to **MailBot** through an OAuth token, only allowing **MailBot** to read emails and view EvanBot's GMail profile. Mailbot **cannot** send emails as EvanBot or delete emails

Mallory is an attacker who wants to exploit the MailBot's OAuth Flow.

Q2.1 (1 point) Select all that are true:

- ☐ The OAuth Token MailBot receives is an example of a capability
- ☐ If MailBot is compromised after receiving access, then the now malicious MailBot can delete EvanBot's emails
- ☐ This is an example of mandatory access control (MAC) because Google enforces the policy centrally
- ☐ Evanbot can revoke MailBot's access without changing their Google password
- ☐ The token follows the principle of least privilege

The following steps occur when EvanBot wants to connect his Gmail to MailBot.

1. **Flow Initiation:** An user clicks "connect" on their client app, which redirects the user's browser to the corresponding authorization server.
2. **Authorization Request:** Authorization server receives user's request along with the requested scopes and a redirect_uri.
3. **User Authentication:** The user logs in and approves access on the authorization server.
4. **Authorization Code Issued:** The authorization server redirects the user's browser to the redirect_uri with a short-lived one time code (auth code: `https://client.com/callback?code=AUTHCODE`)
5. **Code Exchange:** The client exchanges the code directly with the server for an access token.
6. **Account Linking:** The client stores the access token against the currently logged in user.

Q2.2 (1 point) At which step does a vulnerability exist where Mallory is able to inject herself into the process and cause a malicious authorization?

- | | |
|---|---|
| ☐ Step 1: Flow Initiation | ☐ Step 4: Authorization Code Issued |
| ☐ Step 2: Authorization Request | ☐ Step 5: Code Exchange |
| ☐ Step 3: User Authentication | ☐ Step 6: Account Linking |

(Question 2 continued...)

Q2.3 (1 point) What exploit can Mallory do?

Assume:

- Evanbot is currently logged in to MailBot their browser
- Mallory also has a valid MailBot and GMail account
- Authorization codes issued are valid until used
- EvanBot and Mallory are both active on StemEd where users can post and share links
- No State parameters are used in the Auth Flow

Q2.4 (1 point) What underlying attack is this exploit relying on?

- ☐ CSRF ☐ XSS ☐ SQL Injection

Q2.5 (1 point) Assume the attack from above succeeds, what is the end state?

- ☐ Mallory is logged into MailBot as EvanBot and can read all his emails directly
- ☐ EvanBot's MailBot account is linked to Mallory's Gmail
- ☐ Mallory obtained EvanBot's Gmail password through the access token
- ☐ MailBot can send emails on EvanBot's behalf while controlled by Mallory

Q2.6 (1 point) To defend against the attack above, an additional parameter is included in the authentication flow. When a client initiates an OAuth Flow, it will require a state parameter in the authorization request. The state parameter is a random, unguessable value sent along with the request and stored in the current user's session. The authorization server will echo this value back in the callback uri along with the AuthCode.

`https://mailbot.com/callback?code=AUTHCODE&state=RANDOMVALUE`

When the client receives the callback uri, it will first check that the state in the URI matches the value stored in the current user's session. If they don't match, the request is automatically rejected.

Which of the following statements are true?

- ☐ Even with state implemented, the attack would succeed if Mallory could read EvanBot's session data on MailBot's server.
- ☐ The state parameter is the main defense against the attack especially when EvanBot isn't logged in.
- ☐ The state parameter prevents the attack because it encrypts the authorization code in transit
- ☐ If the state parameter is created from Hash(username), then attack earlier is still possible.