Federated Content Filtered Network

Gurupragaash Annasamy Mani
University Of Utah
guru@cs.utah.edu

August 26, 2017

1 Problem Statement

Educational institutions (universities, school districts etc.) often take part in federated authentication via a system called EduRoam. In essence, this allows users of participating institutions with authenticated access to the networks (WiFi/Internet etc.) of member institutions. E.g., students visiting the campus of a participating institution would be authenticated with their home institution and, when successfully authenticated, will be allowed access to the network at the visited institution. This approach has been very successful and is widely deployed on both university campuses and K-12 schools.

This federated authentication approach, however, by design only deals with authentication. Specifically, at a local level, many educational networks apply different network policies to different users, and these policies are not applied in the current federated authentication approach. For example, students of a K-12 school have to be protected against adult content by law on the Web which is ensured by passing their access requests through an appropriate content filtering device. Should a student from one of these schools gain access at, for example, a public library, the network should ensure that this policy is still being applied to the students traffic (while traffic from the general public at the same public library might not be subjected to the same policy treatment). State-of-the-art federated networks authenticate the user across networks but, since the policy is maintained locally in the home system and is not transferred across, the users are granted access to the network without any policy being applied, or with default policy being applied. In some cases, this default group has a strict policies and would satisfy the institution's requirement, but in some cases the user gets access to content which they are not allowed to, leading to breach of FERPA requirements. So to be on a safer side, most institutions do not allow remote users on their network.

We aim to solve this problem by adding a policy exhange phase during the authentication process. At the end of this policy exchange process, if it was successful, the remote institution

can assertain that a correct policy will be applied to the user's traffic. Hence it can allow the user if the credentials are valid, or deny the valid authentication request if the policy exchange was not successful.

2 Naive Solution

There are few straight forward solutions to our problem.

- Send the policy associated with the user to the local network and realize the policy in the local network.
- Have a pre-negotiated policy which can be used for users of different network.
- Tunnel the traffic to the home network.

3 Related Work

There is lot of work in the space which implement the naive solution(sending the policy across networks). PacketFence [2] and Aruba's ClearPass [1] are examples of such systems. These systems are capable of doing authentication across networks, exchange policies across networks, convert the policies and realize them based on the local network's hardware support. But they strongly assume that the policy exchanged can be implemented in the local network.

Instead of sending policies and translating them based on the network support, some of the systems create a higher policy layer which can be understood by all the participant networks. Then the local institutions write policies which adhere to the higher layer. In this way, the policies can be sent across and can be implemented in the local network without any conversion/translation of policies. Multi-OrBAC [4], X-FEDERATE [3] use this approach. There are two main issues with these approaches. 1) Creating a common policy layer and asking all the participants to change implemention to support this policy layer is not feasible and cumbersome where there are lots of participants in a federation. 2) We cannot assume that the policy of one participant can be realized in another participant since the other might not have necessary hardware to implement such a policy. Even if a network has the capacity to instantiate a policy, instatiating one for every user is not scalable.

The issue with the other naive solutions are pretty straight forward. Tunneling the traffic leads to latency and the pre-negotiation of policies across participants is not a feasible task when the participants are huge in number. Due to these reasons, we believe the current solutions do not solve the problems faced in a federated network scenario.

4 Our Solution

Policy is an overloaded term and can mean lots of different things like access control policies, content filter policies, network security policies, data protection policies and so on. For this

work, we will be focussing on content filter policies. As discussed before, the system should not assume that the local network is capable of implementing the policy received from the remote. With that as the main motive, we have built our system, which facilitates a policy negotiation process between the local and remote network using a policy exchange protocol. The policy exchange protocol ensures that at the end of the exchange, both the local and remote network agree on an action which will ensure that the user's traffic is subjected to the policy which is acceptable by the remote network, if not the user will not be allowed on the network.

At the start of the policy exchange protocol, the local network sends its capabilities to the remote network. The remote picks a capability and initiates the protocol associated with the capability. If the protocol exchange results in a success, then this entire policy negotiation process is a success and thus the user is allowed in the network, else the next capability from the received capabilities is picked the process continues till the exchange is successful or the remote runs out of capabilities. In which case, the user is not allowed on the network.

5 Background

Before we go into our system, we need to know 1) How a content filter policy looks like and 2) How federated authentication is performed.

5.1 Content Filter Policies

Figures 1 - 4 shows some of the commonly present configs in the content filtering device. These configs as a whole represent a content filter policy. Figure 1 lets the administrator to specify which categories are allowed for an user, which is blocked, which is allowed but access is logged and so on. Figure 2 lets administrator to control the list of applications the user can have access to. Figure 3 allows administrator to configure fine grained control on social media and finally Figure 4 allows administrator to configure allow lists and block lists. Apart from these generic ones, there are other vendor specific configs. All these configs can be categorized into two types, 1) Fixed Sized Configs and 2) Variable Sized Configs. Fixed Sized Configs are one which has a fixed numbers of key, value pairs and the administrator sets the values for each key. Webcategories, applications and socialmedia are examples of Fixed Sized Configs. Variable Sized Configs are ones which contain a list and the administrator is allowed to add as many entries to the list as needed. Allow Lists and Block Lists are examples of this type of config.

5.2 Federated Authentication

Radius [5] is one of the most widely used systems for authentication in a federated network. We will also be using Radius to do the federated authentication part of the system.

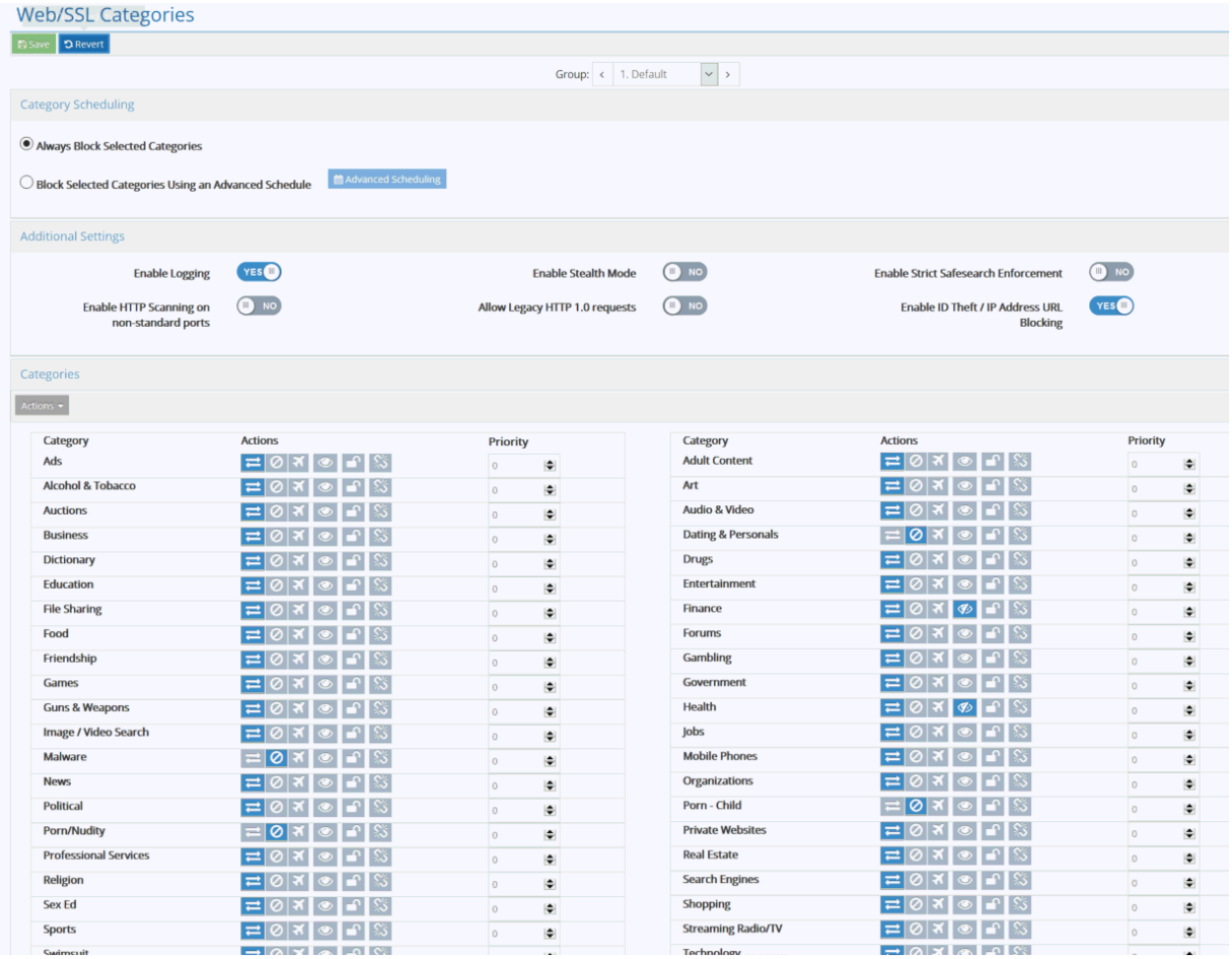


Figure 1: Web Categories

5.2.1 Radius Interaction

The radius interactions are shown in Figure 5. Here the user (phone/laptop) runs a 802.1x supplicant. Any Access Point(Wired/Wireless) where the user connects to is called the Network Access Server(NAS). The radius server which we are using is freeradius 3.0. The NAS is configured as a valid client in the radius server configuration and a shared secret is configured in both the NAS and radius server. Each NAS has its own shared secret with the Radius server. Now when the User connects to the AP(NAS), they use some secure protocol to communicate. It can be EAP, CHAP, MSCHAP, PPP and so on. In this example, we are using EAP. So when the user initiates a EAP Start, the NAS sends EAP request identity message. Now when the user responds with EAP response, the NAS converts this into a RADIUS ACCESS-REQUEST packet and sends it to the radius server after encrypting with the shared key. The radius server decrypts the messages, and gives a challenge to the USER via RADIUS ACCESS-CHALLENGE packet. The NAS converts this ACCESS-CHALLENGE packet to EAP Request Authentication packet and once the user sends a response for this, the NAS converts this to RADIUS ACCESS-CHALLENGE and also adds the

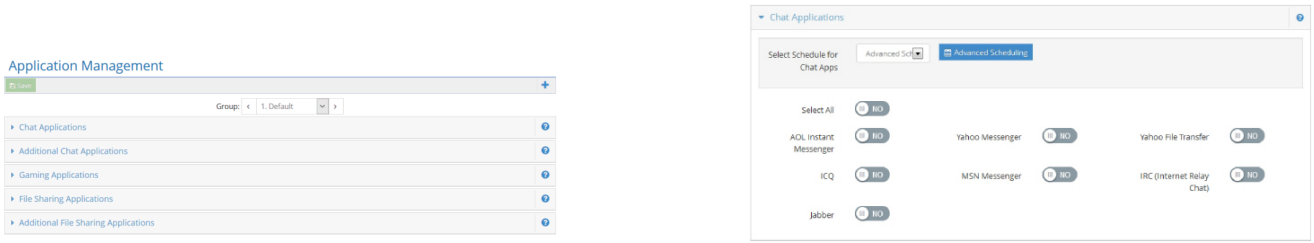


Figure 2: Applications

Figure 3: Social Media

Figure 4: Allow and Block Lists

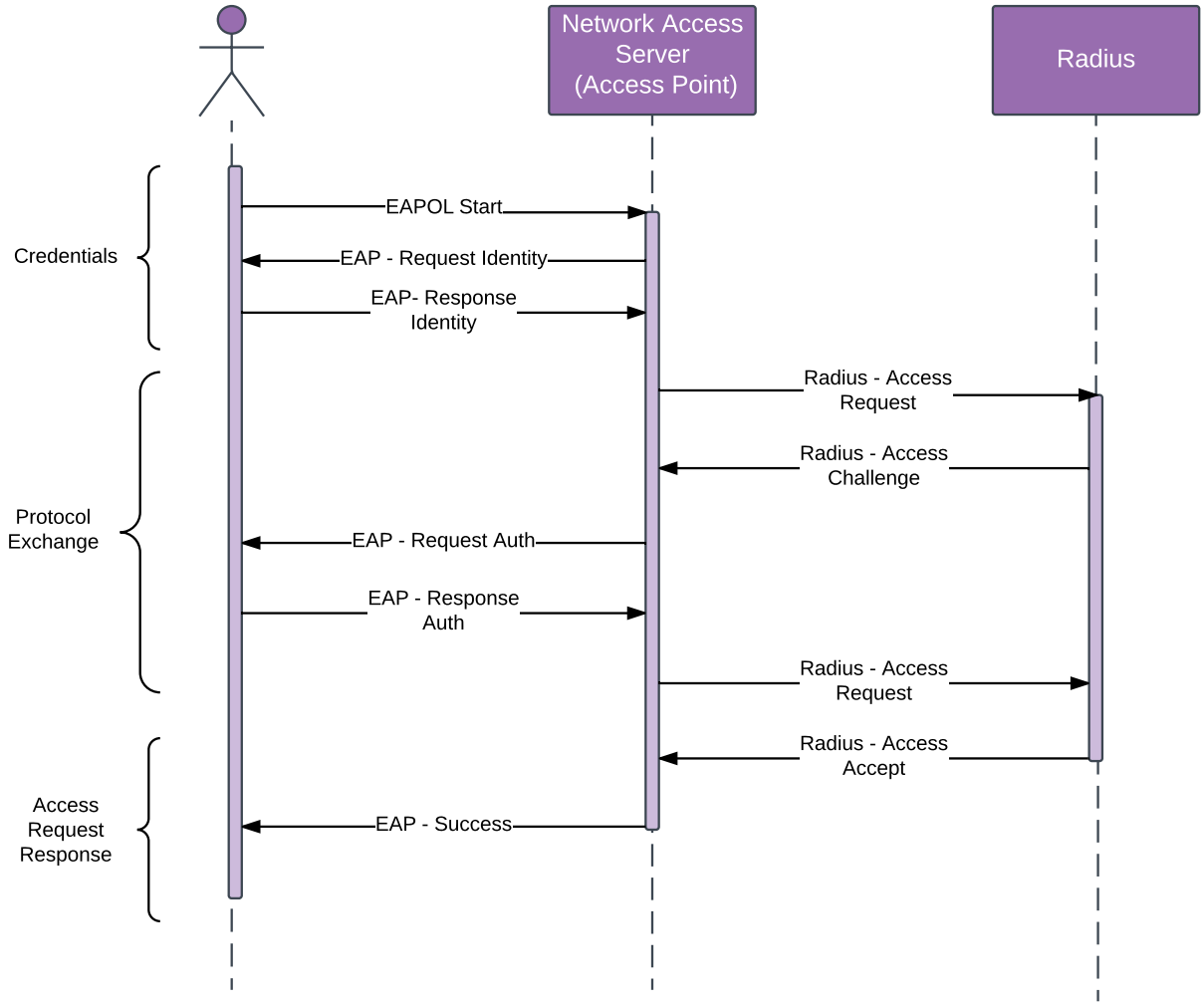


Figure 5: Radius Interactions

state variable to let the RADIUS know it is not a new session. The radius server then processes the response and if its valid will respond with RADIUS ACCEPT-ACCEPT or send a RADIUS ACCEPT-REJECT packet which the NAS will translate accordingly. Now lets consider the case when a user moves across institutions in a federated network. When the user connects to an AP, the users request are sent to the Radius Server configured in the AP. The Radius server looks into the realm attribute of the user. If the realm, is the local realm, then this user credentials will be present in the local DB, else it should pass this information to appropriate Radius Server. Assume there are "N" participants, then each of the participants should have N-1 realm and their Radius Server details configured and whenever a new institution becomes a part of the federated network, all the participants should be updated. This is cumbersome and hence an alternate method is used. All the institutions send the requests to a proxy Radius Server if it is not their realm and this proxy Radius is configured with the realm information and it can

pass this information to the appropriate Radius Server and any changes in the realm will lead to changes only in the proxy Radius Server. There can be multiple proxy servers, hierarchy of proxy servers for realiability and so on. Figure 6 shows the interactions in the proxy case. As shown in the figure, the communication between the Radius Server's are hop by hop and hence in the vanilla Radius protocol, the final Radius Server cannot determine from where the request was originated.

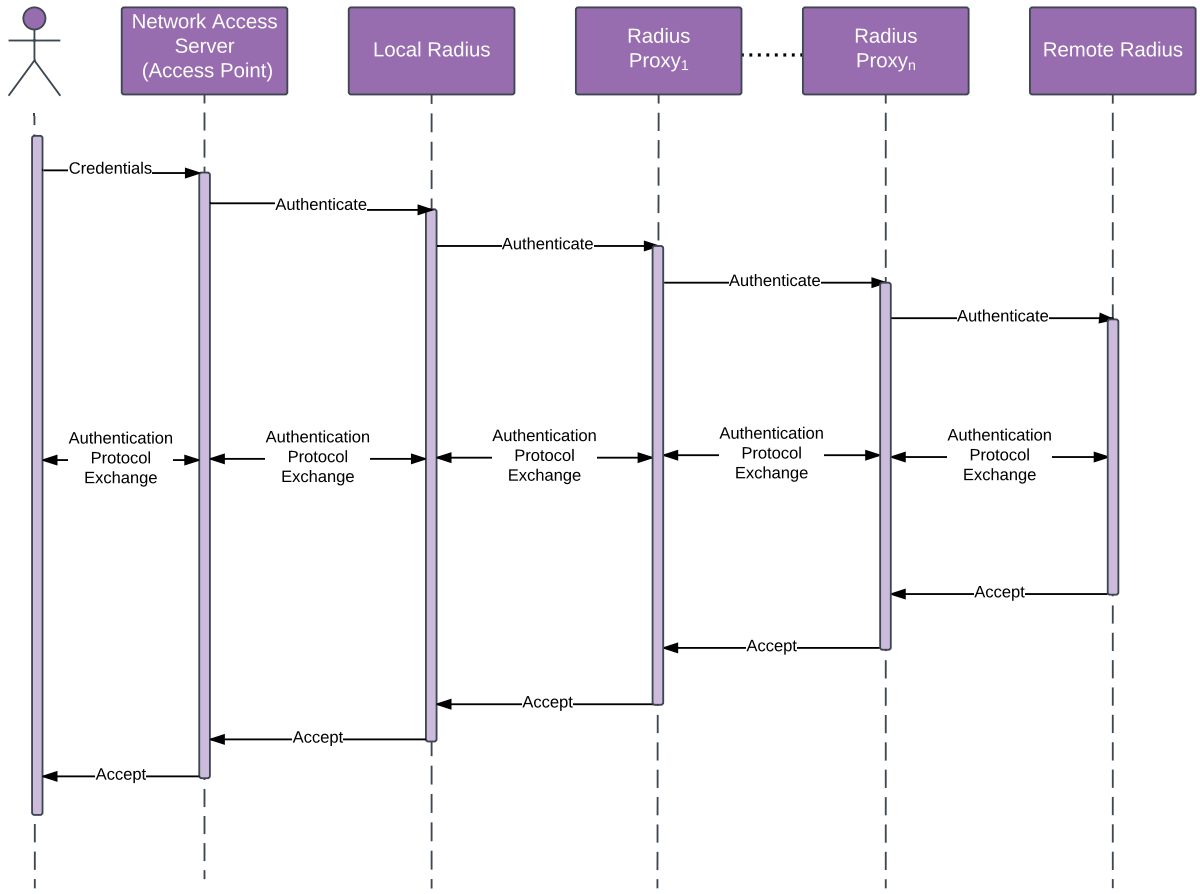


Figure 6: Radius Proxy Interactions

5.2.2 Radius with Policy Exchange

Figure 7 shows where the policy exchange protocol will be hooked in the Radius authentication process. But for this to happen, as shown in the figure, we need to know where the request originated from, so that a policy exchange can be initiated between the appropriate participant networks. We plan to achieve this by using the Proxy-State Attribute in the Radius protocol. According to the Radius RFC2865 [5], a Radius server(proxy) can add any information as a Proxy-State Attribute when forwarding a Access-Request message. The Radius Server receiving the Proxy-State Attribute cannot modify the attribute and if it needs to add its own proxy-state details,

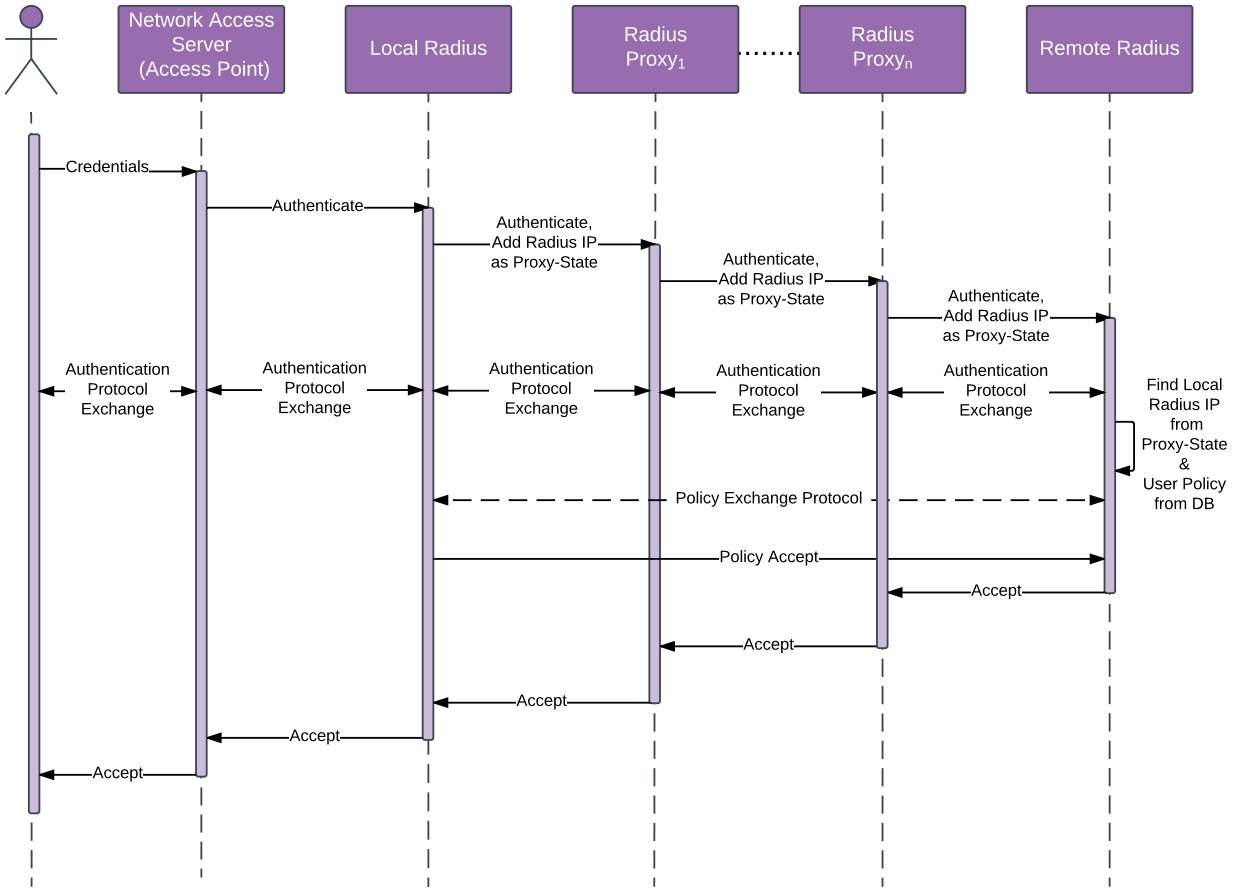


Figure 7: Radius Interaction with Policy Exchange

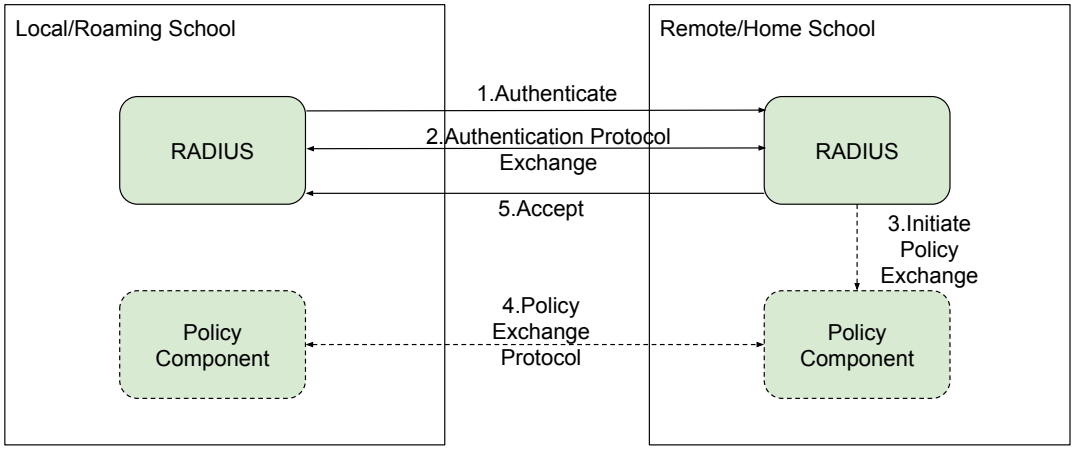


Figure 8: Policy Exchange via Components - Abstract

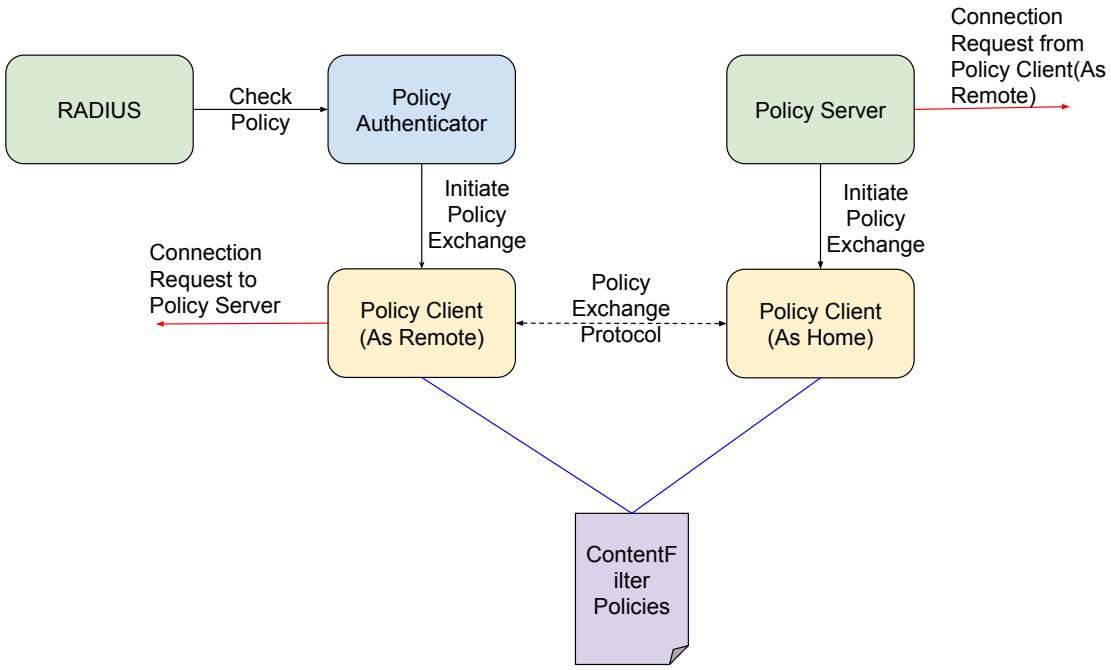
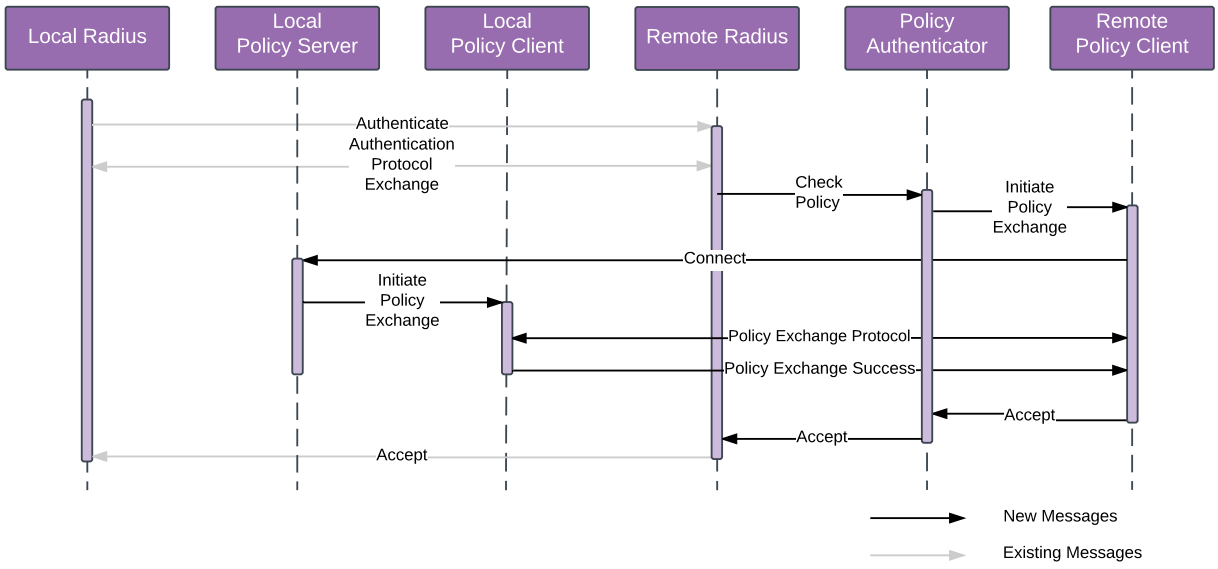


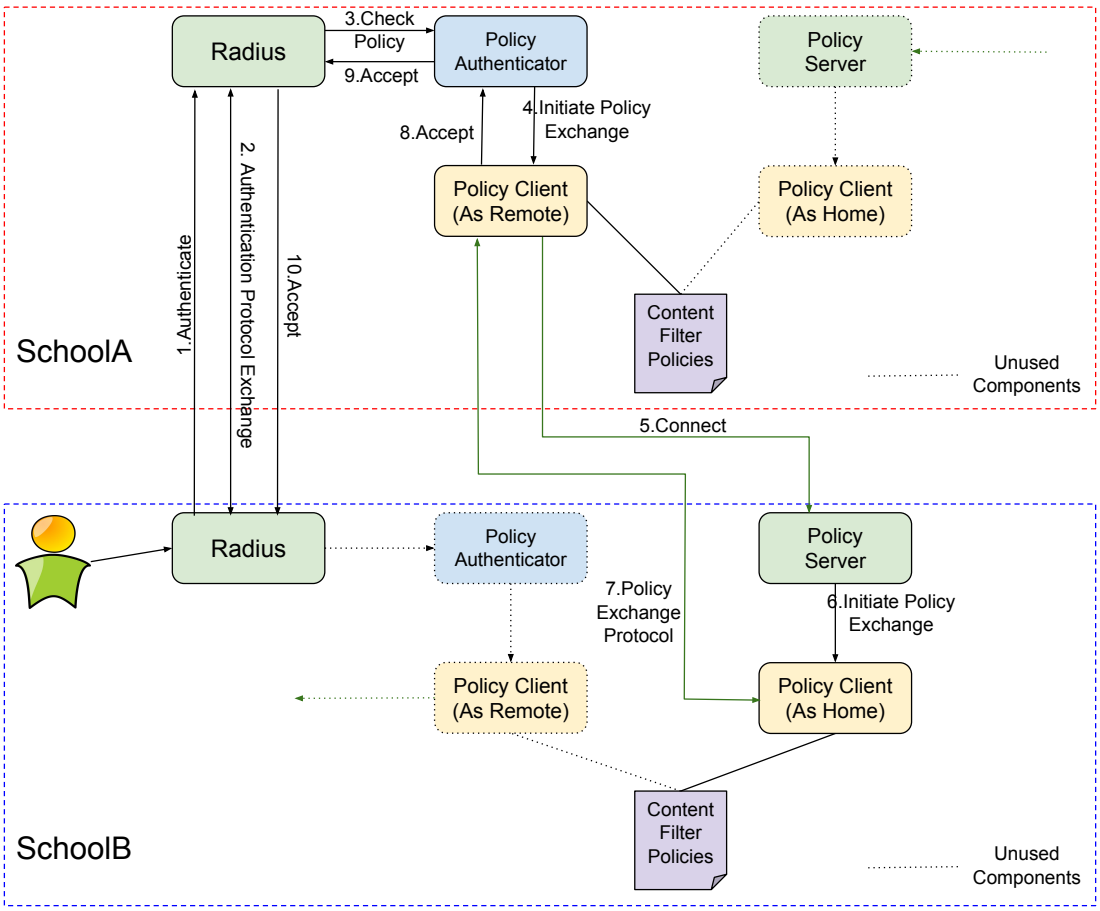
Figure 9: Policy Exchange via Components - HighLevel

it should be added on top of existing proxy-state attribute. Thus the proxy-state attribute added by the first Radius Server will reach the final Radius Server without any change in its value. The Radius Server of the participating instituions will be configured to add their ip address when forwarding an Access-Request packet. Thus the final Radius Server will know where the request was originated from using the proxy-state attribute. If there are multiple proxy-state attributes present, the last proxy-state attribute in the stack will have the origin ip address. Now that we have the origin ip address, we can initiate the policy exchange process, but Radius protocol prevents any server to initiate a communication. According to the RFC, Radius is a request-response protocol and the request should be initiated by the user and hence we cannot do policy exchange using Radius messages. Instead we can create a side channel to do the policy exchange and if the exhange was successfull, we can notify the Radius Server that policy exchange was successful and to proceed with validating the user.

Figure 8 displays the same interactions shown in Figure 7 but in components level. The Policy Component piece is the contribution of this project which enables the policy exchange in the authentication process. The Policy Component piece is comprised of many smaller components which will be discussed in detail in the section 7. Figure 9 shows the same interaction with some of the main components of Policy Component. The Policy Authenticator is a Radius module, which will be hooked to the Radius in the authentication flow. And hence whenever there is an authentication request, Policy Authenticator receives the request and if its from a remote network, it initiates the Policy Exchange process else will reply immediately. The Remote Policy Client performs the policy negotiation with the Local Policy Server via Local Policy Client and the result of this negotiation is passed on to the Policy Authenticator which then informs the



(a) Flow Diagram



(b) Flow via Components

Figure 10: Policy Exchange - HighLevel

Radius Server. Based on the result, the Remote Radius server will decide if its okay to allow the user or not. The Policy Server is a ever-running process which services requests from the Policy Client from the remote network. Figure 10a shows the messages exhanged in this side channel at a high level. Figure 10b shows the same policy interaction we just saw, but also shows what all components are active at each location during the policy exchange.

6 Policy Exchange Protocol

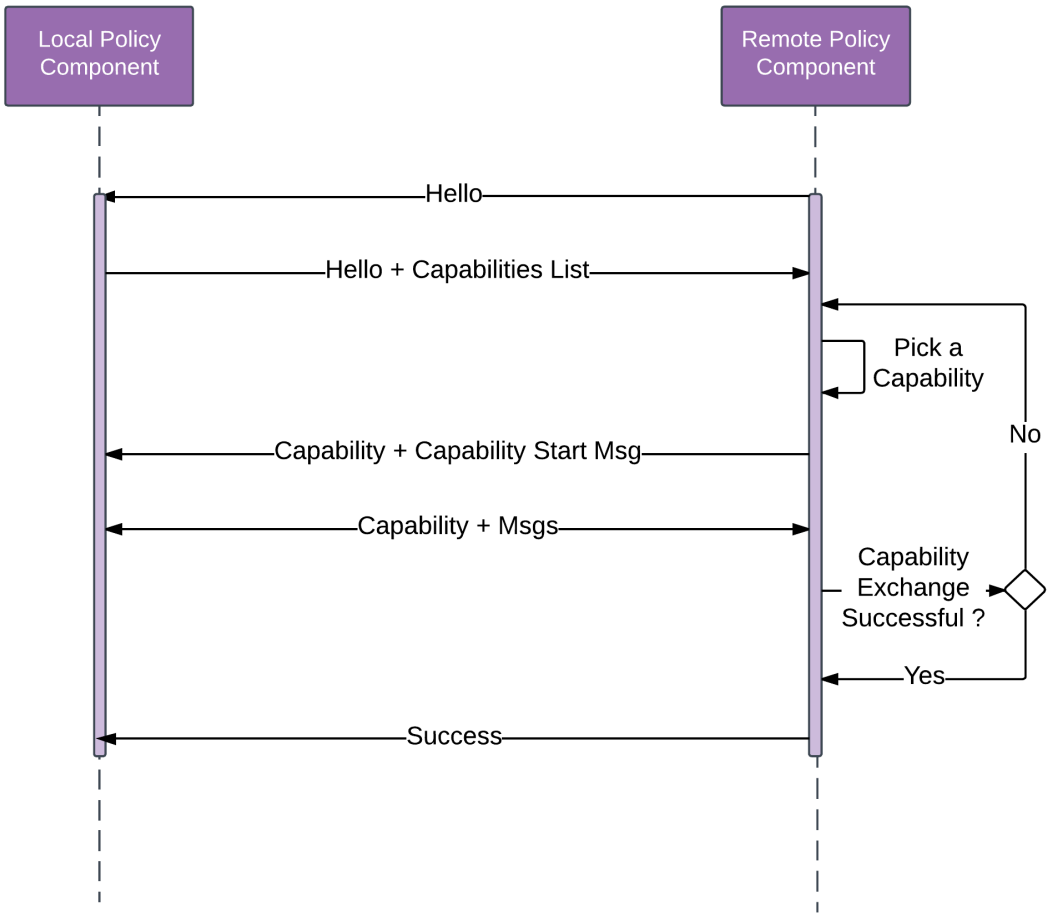


Figure 11: Policy Exchange Protocol

The Policy Exchange Protocol is a sequence of messages sent between the Policy Components of local and remote network. The protocol interactions are as follows

- The Remote sends a Hello message to the Local
- The Local responds with a Hello and a list of capabilities supported by the local network. The Capabilities are discussed in detail in section 6.1.

- The Remote picks a capability from the received list and initiates the exchange associated with the capabilities.
- If the exchange was successful, then it means the Policy Exchange was successful due to this capability and the action associated with the capability will be performed by the local network and the user will be allowed. Since the capability was published by the local network, it will have the resources required to perform the action associated with the capability.
- If the exchange was a failure, the remote picks the next capability in the list and starts again.
- This continues until an exchange is successful or the remote runs out of capabilities. In that latter case, the policy exchange protocol was unable to successfully negotiate policy across local and remote networks, and so the user will not be allowed in the local network.

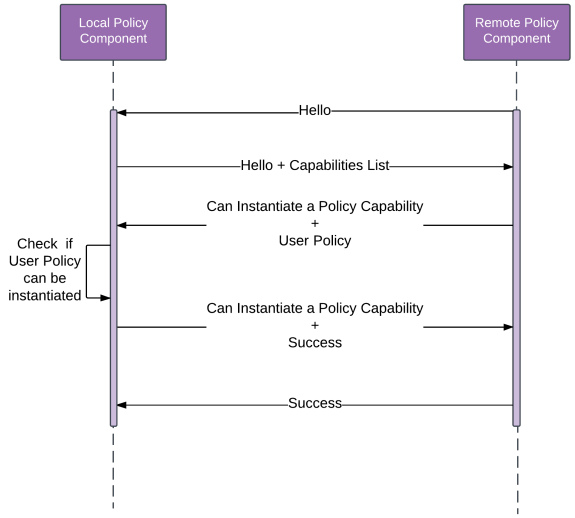
Figure 11 shows the above interactions as a flow diagram.

6.1 CAPABILITIES

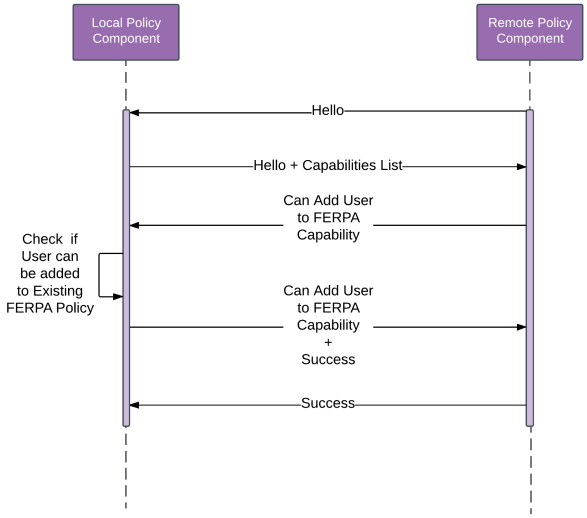
A network can have any combination of the following capabilities

- Can Instantiate a Policy - Given a policy, is the network willing to instantiate this policy. This capability just tells if the network is willing to instantiate or not, it does not guarantee that it will be capable of realizing the policy. Refer Figure 12a for interactions.
- Add user to FERPA Compliant Policy - The network has a policy which is FERPA complaint and can add the user to that policy group. Refer Figure 12b for interactions.
- Tunnel the traffic - Given a destination, is the network capable of tunneling all the users traffic(to whom the authentication is being performed) to the given destination. Refer Figure 12c for interactions.
- Can Negotiate a Policy Capability - Given a policy, this capability aims at finding a policy in the local network which differs from the user's policy in the remote network by a threshold which is acceptable by the remote network. Figure 12d shows the messages involved with this capability and the section 6.2 describes this capability in detail.

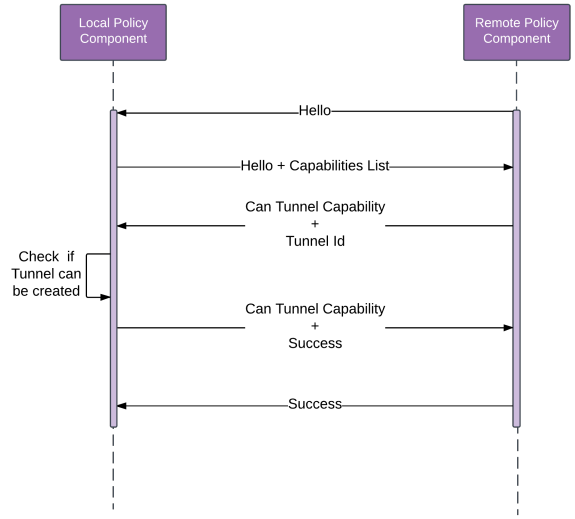
This capability list is not a fixed one. A group of participants could come with a policy, which all of them are comfortable with and this policy could be added to the capability list and when there is a policy exchange between this small group, they can decide on this capability.



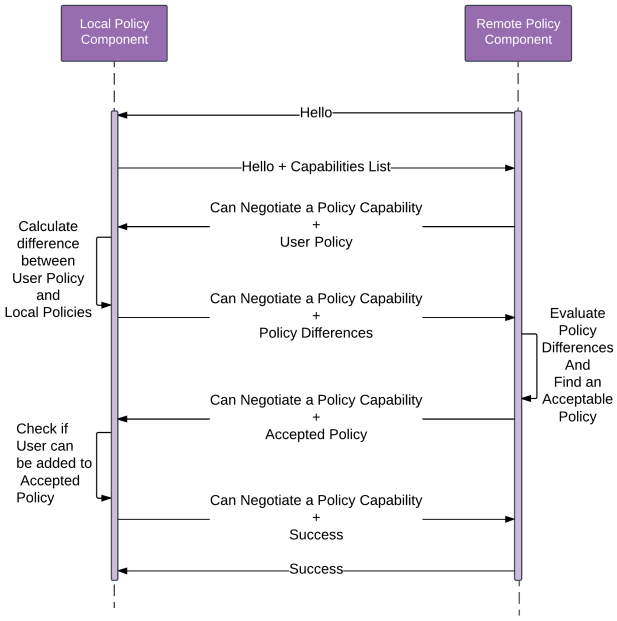
(a) Can Instantiate a Policy Capability



(b) Add user to FERPA Compliant Policy Capability



(c) Tunnel the traffic Capability



(d) Can Negotiate a Policy Capability

Figure 12: Types of Capabilities

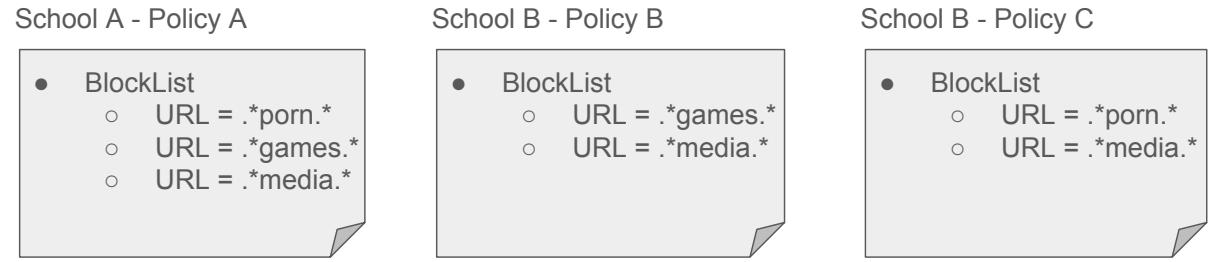
6.2 Policy Negotiation Process (Policy Diff Capability)

This capability aims at finding a policy in the local network which differs from the user's policy in the remote network by a threshold which is acceptable by the remote network. The interactions in this process are as follows:

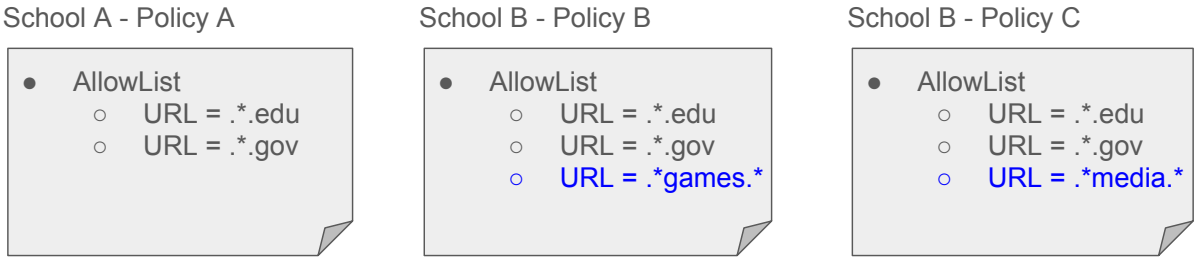
- Along with the 'Can Negotiate a Policy Capability' header, the remote network sends the policy associated with the user($PolU_A$).
- The local network extracts the $PolU_A$ from the message and performs a difference with its local policies. The way to perform the difference is decided based on the type of the content filter policy. Each policy has an approriate differentiator module associated with it and this module performs the difference calculation.
- Once the difference is calculated, the difference is send to the remote network to be evaluated.
- The remote network then evaluates the difference using the appropriate evaluator module. Since each policy can be of different format, the difference can be of different formats. So the appropriate difference evaluator is fetched and evaluates the difference and says which of the given difference is acceptable by the remote network. Section 6.2.1 discusses this evaluation process in detail.
- The remote network then sends the policy which was picked by the difference evaluator($PolU_A^{**}$) to the local network.
- The local network, then checks if the user can be added to policy $PolU_A^{**}$. If so, it returns success to the remote network, else returns a failure.
- Based on the result, the remote will either authenticate the user or proceed with the next capability.

6.2.1 Difference Evaluation

Consider the policy in Figure 13a. In this case, the differentiator will say, that the Policies B and C differ from A by one deletion. A naive evaluator can say, since they both differ by one deletion, they are both are equidistant from the Policy A in similarity measure. But this is without considering the value of the deleted entry. In Policy B's case, a rule which blocks porn sites is removed. Whereas in Policy C, a rule which blocks the gaming sites is removed. The remote school could be fine with its user accessing gaming sites but it need not tolerate allowing porn access to its users. Similarly consider the case Figure 13b, where Policies B and C differ from A by one addition. Eventhough they both differ by one addition, the remote school can prefer Policy B since it only allows gaming sites and not Policy C which allows media sites. So in both the cases, if there was an administrator looking at the policies, they would rate Policy B closer to Policy A instead of Policy C in both the cases. We want to achieve the same in our system but without the help of an administrator at runtime. Hence we have introduced a component



(a) Policies with an entry, missing



(b) Policies with an entry, added

Figure 13: Sample Policies

called Acceptance Policy, which the administrator can use to configure the network's preference on how the difference should be evaluated. This is a one time configuration activity and once the Acceptance Policy is created, then the Policy Evaluator will use this Acceptance Policy while evaluating the differences and find which Policy is acceptable by the network.

6.2.2 Acceptance Policy

Figure 14b shows the syntax of the Acceptance Policy. <Config File> field specifies what type of config it is, like AllowList, BlockList, Categories, Keywords and so on. The Operations can be Addition/Deletion. Each entry under an operation, consists of 4 values. Field contains the value, which refers to the field in the difference, this rule should be matched on. In our example its the URL field in the difference. The value field contains the regexp which should be applied on the value of the matched field. If the regexp matches the value, then this rule is a hit and the appropriate penalty value should be added to the total penalty in the policy evaluation process. The priority determines what is the priority of this acceptance rule. The lower the priority value, the higher its priority. This helps us to solve conflict when two acceptance policy rules matches a difference. If two rules with the same priority matches a difference, the rule with higher penalty takes the preference. Now lets apply the sample acceptance policy shown in Figure 14a over the policies in 13a. The Difference Evaluator will say Policy B differs Policy A by a penalty of 1000 whereas Policy C differs Policy A by a penalty of 10. Thus with the help of Acceptance Policy,

- BlockList
  - Operation - Delete
    - Field - URL, Value - .*porn.*, penalty - 1000, priority - 1
    - Field - URL, Value - .*games.*, penalty - 10, priority - 1
    - Field - URL, Value - .*media.*, penalty - 10, priority - 1
  - Operation - Add
    - Field - URL, Value - .*, penalty - 1, priority - 1
- AllowList
  - Operation - Add
    - Field - URL, Value - .*porn.*, penalty - 1000, priority - 1
    - Field - URL, Value - .*games.*, penalty - 10, priority - 1
    - Field - URL, Value - .*media.*, penalty - 10, priority - 1
  - Operation - Delete
    - Field - URL, Value - .*, penalty - 1, priority - 1

(a) Sample Acceptance Policy

- <Config File1>
  - <Operation1>
    - Field - <Xpath>, Value - <regexp>, penalty - <value>, priority - <value>
    - ……
    - ……
  - <Operation2>
    - ……
    - ……
  - ……
- <Config File2>
  - ……
    - ……
    - ……
  - ……
    - ……
    - ……
- ……

(b) Acceptance Policy Syntax

Figure 14: Acceptance Policy

Difference Evaluator can evaluate the differences in a way personalized to the network.

7 Policy Components

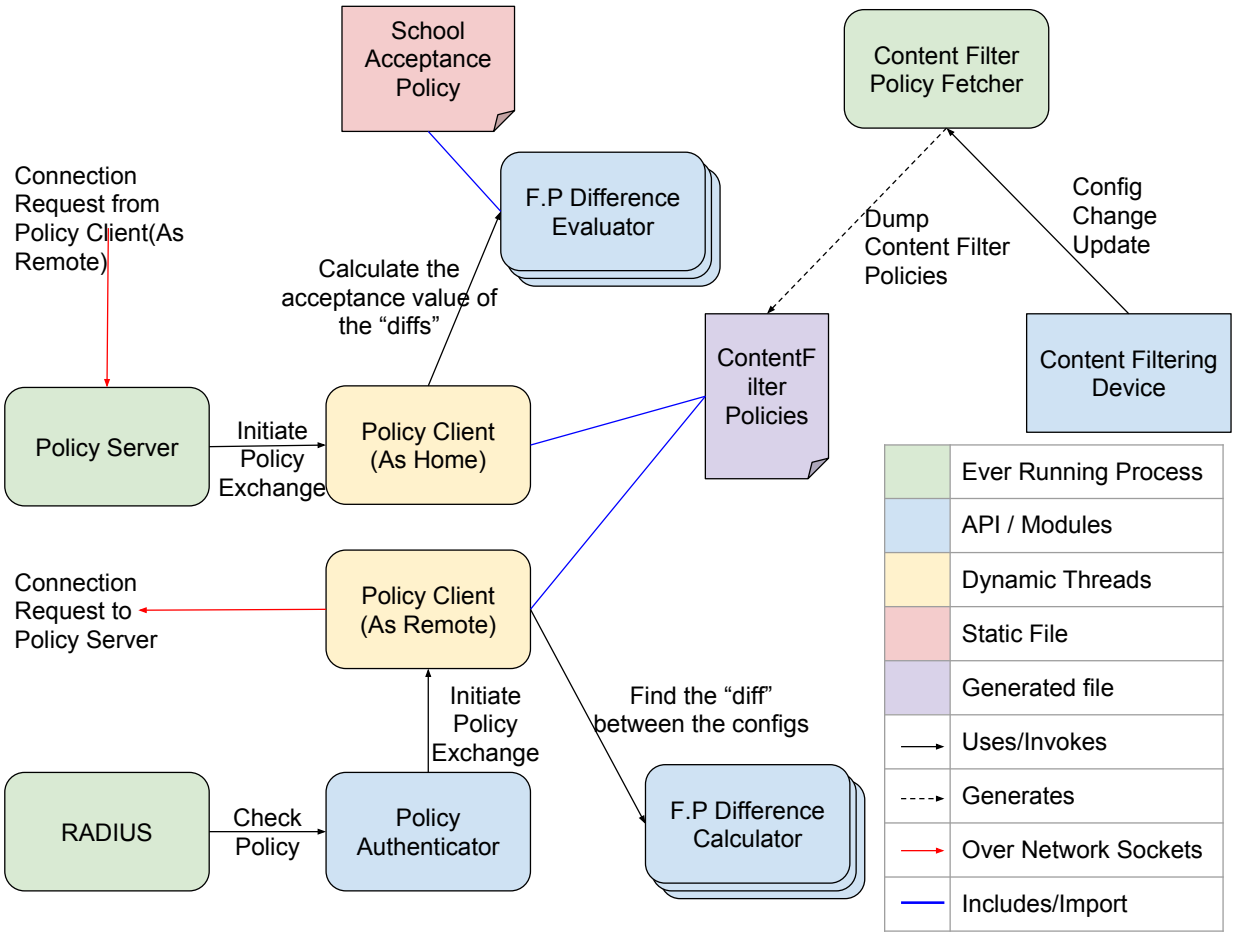


Figure 15: Policy Components and their Interactions

Figure 15 shows the all the components of the Policy Exchange Framework. The components are explained in detail in the sections below.

7.1 Content Filter Policy Fetcher

It is a process which monitors the content filtering device. Whenever the configs in the device are modified, it fetches the updated configs and updates the cache (Content Filter Policies). Instead of making this a process and continuously monitoring the device, the device can be configured to trigger this process whenever there is an update on the device's config.

7.2 Content Filter Policies

This is the cache of filter configs configured in the content filter device.

7.3 Policy Difference Calculator

Given two Content Filter Policies, this module calculates the difference (diff) between the two policies. Since there can be different types of content filtering device, their policies will have varying format and hence this module will also differ based on the type of policy.

7.4 Acceptance Policy

This represents the network's high level policy regarding content filtering. It has details regarding which of the content filter configs are mandatory ones(no social media, no porn) and which are not mandatory but preferred (no games, no sports). There is penalty values associated with each of the preferred configs. Higher the value the more preferred it is. The mandatory configs will have the highest penalty value possible in the system.

7.5 Policy Difference Evaluator

This module works on the policy diff provided by the appropriate Policy Difference Calculator. It looks into the policy diff and evaluates the diff based on the acceptance policy. At the end of the evaluation process, if this policy diff's penalty value is greater than the acceptance threshold of the network, it is not in accordance with the Acceptance Policy, else it is preferred by the network but with a penalty score associated with this difference. At the end of all the difference evaluation, returns the policy with the lowest penalty score as the negotiated policy.

7.6 Policy Authenticator

This is the module which will be hooked to the Radius' User Authentication flow. So whenever an authentication request from the user is posted to the Radius server, this module will be triggered. If the authentication request was from the local network, then the module will return immediately with a success message. If the authentication request was from a remote network, it will fetch the Policy Server Ip of the source network from the Proxy State Attribute in the request message, and spawns the Policy Client process in Remote mode. When the Policy Client returns, based on its return value, will either reply success or failure. If this module's reply is a failure, Radius will fail the user's authentication request.

7.7 Policy Client(As Remote)

This process is spawn by the Policy Authenticator, whenever it receives authentication from a remote network for a local user. The process is instantiated with the user details and the Ip

Address of the local network's Policy Server. It initiates the policy exchange protocol with the local network's policy server

7.8 Policy Server

A server listening on a particular port, waiting for connections from Policy Client(Remote Mode). When a connection request is received, accepts the connection, spawns a Policy Client in Local Mode and passes the connection handle. It then goes back to listen for connections while the Policy Clients communicate within themselves.

7.9 Policy Client(As Local)

This process is spawn by the Policy Server upon request from a Remote Policy Client. Once spawn, it parses the message queue to see if the policy exchange protocol was initiated by the remote policy client. If so, it starts to perform the policy exchange protocol with the remote policy client.

8 Implementation

Current implementation supports only one vendor iboss. Iboss stores all its configurations files in the form of json. And hence the Policy Differentiator and Policy Evaluator modules are tailored to work on json objects. Have implemented the system in python and the code is available in https://gitlab.flux.utah.edu/guru/content-filter. Please utilize README and the comments in the code base to navigate the code.

8.1 Changes in Radius

1. Add 'Policy Server Ip' in the proxy state attribute. Since we want to add 'Policy Server Ip' only for proxied requests, we will use the pre-proxy section. Enable the files option under pre-proxy section in both the default and inner-tunnel config file. Then in 'mods-config/files/pre-proxy' file, add the below config

```
DEFAULT [Realm == 'test.com']
  Proxy-State += '%{Policy-Server-Ip}'
```

   The '+=' will add a new proxy-state attribute in the request with the 'Policy Server Ip'. If the 'Realm' value is set to some domain name, then the proxy state is set only when requests are proxied to that particular server. This will be helpful, if we want to maintain different policy server for different realms, else this field can be ignored and the 'proxy-state' attribute will be set for all the proxied requests. If the proxy-state is being currently used for different purpose, we will have to add a delimiter(id) in the values and based on the delimiter the processing of proxy-state should be determined.

2. To enable the python module in the radius authentication process, follow the steps present in `https://wiki.freeradius.org/modules/Rlm_python`. Then instead of the default example.py, use the example.py from the git.

3. To add the python module in the authentication process, in the authorize section present in sites-enabled/default, add a line 'python' at the end. If there is already a python module present in the middle of the authorize section, comment that line. We should have the python module at the end because, we want the policy negotiation to start only if the user is successfully authenticated.

9 EVALUATION

If the system should be deployed, it should be able to negotiate the policies before the authentication times out. Default Radius Authentication timeout is 30 seconds(can be extended upto 300 seconds) and hence our negotiation process should complete within 30-300 seconds. In our system, only the evaluation of differences is a significant operation since it depends on two variables - difference size and acceptance policy size. The policy difference size is directly dependent on the Variable Sized Configs like allowList and blockList. Hence for this evaluation, we calculated the time taken by policy evaluator for varying difference size and acceptance policy size. We got actual content filtering policies used by a school which had around 60 allowList, blockList combined. Using these entries and fortune 500 company urls, we created 2800 policies whose allowList,blockList size varied from 1 to 560. So the difference between the policies can vary between 0 to 560 entries. Then we came up with 4 acceptance policies with rule sizes 2, 4, 8 and 16. We then ran the difference evaluation across all the polcies for different acceptance policies. The result of the experiment is shown in figure 16. From the figure, we can find that as the policy difference size increases, the increase in time taken is linear. Similarly for a given difference, increase in the acceptance policies leads to linear increase in the time taken. Its only when both the difference and acceptance policy size increases the time taken increases exponentially. In our experiment the maximum time was taken for 506-560 differences with 16 acceptance rules is around 0.45 seconds which is well below 30 seconds. And hence deployment of our system is feasible.

10 CONTRIBUTION

- Adding a capability based policy exchange process during the authentication phase.
- Instead of assuming policies can be instantiated in the local network, a policy negotiation process which finds policy similar to user's policy in the local network.
- An implementation and evaluation which shows that our system is feasible and can be deployed in real time networks.

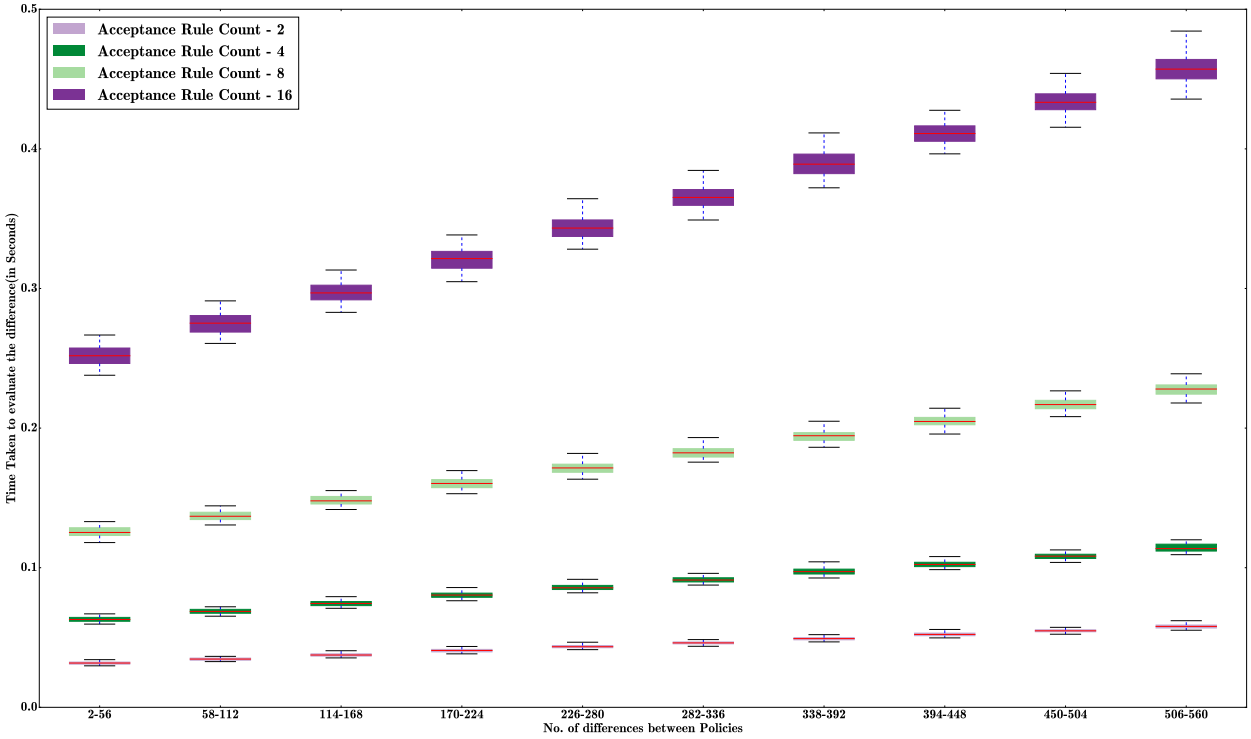


Figure 16: Evaluation Results

11 Future Work

Should add support for multiple vendors. This can be done in two ways

- Add a driver layer for each vendor which can understand other vendors and convert the policies.
- Add a driver layer which will convert a vendor config to vendor agnostic format and vice versa. This can lead to loosing granular configs during the conversion.

12 Acknowledgement

I thank my advisor Jacobus Van Der Merwe for guiding me throughout the Master's program, without whom this project would not have been possible. I also thank Joseph R Breen III and Pete Kruckenberg for their ideas, suggestions and feedback which helped me a lot towards this project.

This project is based upon work supported by the National Science Foundation under award #1642158

REFERENCES

[1] Aruba clearpass - an overview. http://www.arubanetworks.com/assets/so/SO_ClearPass.pdf, 2017.

[2] Packetfence - an overview. https://packetfence.org/about.html#/overview, 2017.

[3] BHATTI, R., BERTINO, E., AND GHAFOOR, A. X-federate: a policy engineering framework for federated access management. *IEEE Transactions on Software Engineering 32*, 5 (2006), 330–346.

[4] EL KALAM, A. A., AND DESWARTE, Y. Multi-orbac: A new access control model for distributed, heterogeneous and collaborative systems. In *8th IEEE International Symposium on Systems and Information Security* (2006), p. 1.

[5] RIGNEY, C., AND WILLENS, S. Remote Authentication Dial In User Service (RADIUS). RFC 2865, June 2000.