Table of Contents

Using Edit Distance to Reduce Bad Realm Proxying

Summary

A way to reduce the proxying of 'Bad Realms' which reduces the reliance on site administrators to add filters and keep their systems up to date.

Problem

Jisc publishes documentation[1] to help sites filter out inappropriate realms with the aim of reducing the number proxied to the NRPS and to help sites spot client device configuration problems. The documentation suggests filters for syntactically invalid strings (e.g. camford..uk), misspelled string (e.g. cmford.ac.uk) and random other common invalid realms (e.g. hotmail.com).

The first and the last are easily filtered out with static filters on each RADIUS server but the second, the misspellings, needs attention from system admistrators to continually update their list of incorrect realms. Logs are provided by eduroam listing such realms and reminders are sent to sites which send large amounts but it's a manual process and administrators have other responsibilities.

Govroam administrators don't appear to be engaging the requests to filter. Whether this is lack of time, resources or ability is unknown but if there is a way to take this away from them then it could be of benefit.

Edit Distances

Spell checkers use algorithms to calculate which word it thinks should have been typed and offers suggestions of similar words. They work by calculating the 'edit distance' (or Levenshtein Distance[2]) between the typed word and every other word in the dictionary. The closest matches are suggested. An 'edit distance' can be thought of simply as the number of edits required to a string to turn it into the other string. e.g. 'place' to 'plaice' requires one addition (edit distance of 1), 'cat' to 'dog' requires three substitutions (edit distance of 3), 'computer' to 'compute' requires one deletion (edit distance of 1).

1 https://community.jisc.ac.uk/library/janet-services-documentation/filtering-invalid-realms
2 https://www.cuelogic.com/blog/the-levenshtein-algorithm/

This approach could be used to calculate a metric for how close a received realm string is to, say, the local realm for a site e.g. for 'manchester.ac.uk'. See Table 1 and Appendix 1

A distance of 0 means that the strings are identical, a low value means that there's a high liklihood of a misspelling and a high value indicates that the strings aren't similar at all.

Realm	Distance
manchester.ac.uk	0
mnchester.ac.uk	1
manchesterac.uk	1
manchester.co.uk	2
manchester.ac.co.uk	3
manchester	6
student.ac.uk	6
hotmail.co.uk	7

Table 1: Example distances

Algorithm

If a RADIUS server can implement some decision logic then it could work like this:

1. For each incoming RADIUS request compare the realm part of the Username with the local site realm.
2. If the edit distance is 0 then it's a known realm, deal with it appropriately,
3. If the edit distance is below a defined threshold (say 3) then the realm is a misspelling of the local realm by a local user and send back a reject,
4. For all other (higher) edit distances, assume that it's a different realm entirely and proxy to the NRPS/next level in hierarchy.

In practice, point 2 wouldn't be needed because the basic RADIUS configuration should handle matching all the local realms anyway.

Implementation

There are several critical elements to implemeting this approach:

- There must not be a significant overhead to processing these comparisions. Many sites have to process many requests per second and delays could cripple a service.
- If in doubt, the RADIUS server should proxy the request. Thus any defaults within the comparison software, RADIUS configuration or other must assume this.
- Timeouts should be short. RADIUS servers should not have to wait for any significant length of time for processes to time out.

Potential implementations:

- A compiled program that could be distributed via Github for a variety of platforms (Linux

32/64 bit, Windows 32/64bit). Supplied parameters would be two string and an integer for the site realm, submitted realm and threshold. Return value would be the edit distance.

- Modules for RADIUS servers such as FreeRADIUS and RADIATOR.

Sites would integrate this software into their RADIUS servers software along with the configuration to make suitable decisions about the response/proxy. The threshold can be decided by the site. It should be possible to run it in line but logging the potential decisions, rather than enacting them. This way sites would be able to benchmark different threshold choices.

Running individual binaries for each request is less efficient than, say, a module built specifically for FreeRADIUS. System resources are required to start up a process but not for build-in modules. Whether or not it's possible to integrate with Microsoft NPS, Cisco ISE as binaries or modules is unknown at this point.

Enhancements

Algorithm/Implementation

It is worth noting that the Levenshtein algorithm isn't particularly efficient. The number of operations is a product of the length of each string. It is not recommended for long strings but the the strings used for realms are mostly less than 32 characters.

Our requirements isn't to calculate the edit distances between two strings, but, rather, to determine if the edit distance is greater than some threshold.

Neither does the implement have to be 100% accurate. As long as the program errs on the side of caution (telling the RADIUS server to *proxy* rather than *reject*) then it's doing a good enough job (reducing, rather than eliminating, bad realms).

Obvious enhancements and time saving mechanisms:

- If the strings are an exact match then don't perform the comparison, just send to the IdP.
- If the strings lengths differ by more than the threshold value, then it's safe to assume that the edit distance is exceeded, don't perform the comparison, just Proxy.
- If either of the strings is greater than, say, 20 characters, then don't perform the comparison, just *proxy*, to save processing time.
- Design the software to exit with a '*Proxy*' message if the execution time exceeds a limit (say, 10ms).

Added benefits

The most basic use of the software is comparing two strings and determining if they exceed a threshold. However, it could do much more. Rather than expecting sites to implement the realm filtering rules as mentioned in the at the start, it would be possible to emcompass these into the software. Checks for realm syntax and matching a list of strings are easy and very low cost, in terms of processing.

Why it won't work

This all sounds good, right? Well, unfortunately, as described above, it won't work, particularly not for eduroam. The whole system relies on the visited site realm to be the only realm that's within, say, three edit distances of the requested realm. Unfortunately, this just isn't the case. Eduroam's realms may be 8-12 characters long but 95+% of them end in the six characters '.ac.uk'. There are plenty of realms with only two preceeding characters. Analysis (see Appendix 2) gives 130 realms which are just one edit distance apart and 1,500+ two apart.

Thus if someone from tmc.ac.uk turns up at a wmc.ac.uk site they'd be rejected if a threshold of '2' was being used. As would anyone from ic.ac.uk, qmu.ac.uk, rvc.ac.uk etc.

The problem isn't as prominent with Govroam (see Appendix 3) because a) fewer realms (146 v. 841) and b) more variation in name space (.nhs.uk, .gov.uk, .org, .net etc.). Sites with similar names are also more likely to be found within Federations. However, over time there will be more and the chances of clashes happening will increase. A threshold of one, currently, wouldn't cause a problem but a threshold of two would.

So what's the point of all this?

All Realm Edit Distance Checking

What if it was possible to check every authentication request against every known realm? Then the aforementioned problem wouldn't exist because, as the algorithm above demonstrates: step 1: if there's a match then the RADIUS server will know what to do with a known realm.

There are ways that this could be achieved

1. Distribute the software described above, but, rather than just matching against the local realm(s), check **all** realms (or, at least, all UK realms). The software would have to be able to download the latest list of realms on a regular basis. This approach would have some practical limitations.
2. Have a central service that does the checking. Using a client/server model, send a query from the RADIUS servers to Jisc which performs the checks and responds with the edit distance.
3. A combination of the above where local software performs queries to a central server

then caches the results (and/or a list of all domains) against which it can query subsequently. Appropriate expiry times would be needed for the entries.

Performance

The most obvious concern would be around performance. A central service would, potentially, need to respond to **every** authentication request for eduroam/Govroam in the UK. Some Universities are handling 30-40 authentication requests **per seond** at peak times. As stated above the edit distance comparision algorithms aren't the most efficient. Comparisons of two 16 character strings are quick but comparing a string to 800+ other strings would be much slower.

Remember though, that the implemention could be optimised and have suitable time outs. As long as the client software was written to respond to the RADIUS server quickly and err on the side of 'proxy' then at peak times the sites shouldn't suffer any issues.

It's also worth understanding that, while there are a lot of misspelled realm requests, the number of distinct misspelled names is much, much lower due to the way that clients retry and retry after a reject. Thus caching at both the client and server ends would massively reduce the query times and loadings.

The first time a query for a particular realm is made the process would be:

1. RADIUS server runs client with requested realm, local realm(s) and threshold.
2. If the edit distance is 0, the client returns '*local*' immediately
    - RADIUS server handles the authentication locally
3. If the edit distance is greater than the threshold, the client returns the value '*unknown*' immediately
    - RADIUS server proxies to the NRPS
4. Otherwise the client sends a query to the central server with the requested realm.
5. The central server compares the realm with all the known realms
6. If the edit distance is 0, returns '*known*' to the client.
    - Clients returns '*remote*' to the RADIUS server (which then will proxy the request)
7. If the edit distance is greater than a central threshold, returns '*unknown*' to the client and caches the result for that realm and sets the expiry for 24 hours.
    - Client returns '*unknown*' to the RADIUS server (which then will proxy the request)
8. Otherwise returns '*mistake*' to the client and caches the result for that realm and sets the expiry for 24 hours.
    - Client returns '*mistake*' to the RADIUS server which will then reject it

Then the next time there is a request from for that realm from that site, or any other:

1. RADIUS server runs client with requested realm, local realm(s) and threshold.
2. If the edit distance is 0, the client returns '*local*' immediately
    - RADIUS server handles the authentication locally
3. If the edit distance is greater than the threshold, the client returns the value '*unknown*' immediately,
    - RADIUS server proxies to the NRPS
4. Otherwise the client sends a query to the central server with the requested realm.
5. If the edit distance is 0, returns '*known*' to the client.
    - Clients returns '*remote*' to the RADIUS server (which then will proxy the request)

Worst case a realm such as 'manchwster.ac.uk' being used from the 'manchester.ac.uk' site for the first time. That would require a local lookup, then passed onto the central server, which would perform a full comparison.

It would be reasonable to assume that the misconfigured client with 'manchwster.ac.uk' would continue to be used at the manchester.ac.uk site so the result would be cached after that first attempt. Only once every, say, 24 hours would the full lookup processing be done for the realm.

The central server list of valid realms would need updated every time a new realm is added or removed but this is relatively infrequent.

Software Design

The software would need to be written specially either from scratch, or using a high performance client server framework such as gRPC[3] (see Appendix 4). The benefits of writing from scratch are that it can be made as specialised as possible (UDP transport for speed and lack of connections to hang) but would take longer to become reliable. Software such as gRPC provides an existing well-tested bedrock to build on and would make development much faster.

Another option might be to see if it's possible to convert existing software, such as DNS software to this purpose. DNS is a low latency, UDP protocol with caching, performs lookups etc. It's already a close match to what's required.

Enhancements

- It might be possible to use DNS queries to help with the realm checks. DNS lookups are very fast, low latency, UDP and there are caching servers. In the case of eduroam (less so for Govroam) virtually all the realms should match DNS entries.

3 https://grpc.io/

- For larger sites installing the server software locally could add an extra layer of caching. i.e. the client queries a local server, which has results cached. If the local server doesn't know the answer then it'll query the central server and cache the results. The local server wouldn't have the same list of all realms that the central server would have. The local and central server software could be identical, but started with different options (e.g. cacheonly v. loadrealms).
- For performance, the servers could return '*proxy*' immediately for the first query about a misspelled realm, and then in the 'background' perform the comparison at their leisure and populate the cache for the next such query for the realm.
- Modules written for RADIUS servers such as FreeRADIUS could include their own local caches, removing the need for running the caching servers locally.

Failures

All the software involved should be written or configured deal quickly with failures. The RADIUS server should default to proxying if there's no/slow response from the local client, which in turn should send back a '*proxy*' response if there's no/slow response from the central server. The central server should default to shutting down rather than causing delays.

Added Benefits

As suggested in the previous Added Benefits section, this checking system could take over the responsibility for checking invalid syntax and for 'hotmail' type realms. Jisc could take responsibility for defining a suitable set of filtering rules that would be instantly applied at all sites.

Jisc (and the eduroam/Govroam) communities could benefit from the information gained. Analysis of logs and the cache could give more information about realm usage and mis-configurations.

Testing

Once a suitable piece of software as been written then it needs testing in an environment in which it can be thoroughly evaluated. Jisc's own ORPS, the University of Manchester or the University of Loughborough might be candidates?

Further Discussion

Algorithms

The Levenshtein algorithm isn't the only way to calculate edit differences.[4] There are simpler

4 https://en.wikipedia.org/wiki/String_metric

ones, and more complex ones. These may return better results and should be considered as alternatives. There are costs (in processing time) to more complex algorithms. However, simpler algorithms may be less accurate. There should be a comparison done between the various algorithms, using real data, to determine the cost/benefits of them.

Software

The response value from the software is up for discussion. It could supply a series of responses such as '*doubledot*', '*doubleat*', '*blacklistedrealm*', '*exceedsthreshold*', '*stringtoolong*' and let the RADIUS server process these accordingly. Alternatively returning a simpler *proxy/reject/local* set of instructions might be better.

Caveats

The smart question right now would be, what about international realms? The initial proposal wouldn't work because there would be similar enough realms to trigger a reject inappropriately. What if there are international realms that would so clash with UK ones. i.e. if someone from manchester.ac.es visited manchester.ac.uk then a threshold of 3 would see their request being rejected.

Further analysis, using the entire eduroam realm data set would be required but my guess would be that the change in TLD would be enough to mean that a threshold of 2 would be guaranteed to work and 3 highly likely to do so.

Final Summary

This system should be seen as an enhancement to the Federated Roaming system. It's not a dependency and no part of the Roaming system should suffer due to lack of availability of any part of the service. If it performs as expected then it could reduce the load on the NRPS (and Govroam RRPS), make the administration of ORPS easier for the administrators, provide a more consistent realm filtering configuration and allow Jisc to manage the filtering configuration centrally.

Software needs to be written (or repurposed) and tested in real situations.

Appendix 1: Analysis of Manchester data

Manchester's ORPS are configured with a list of misspelled realms to reject. Using these as the source data they were compared to the realms 'manchester.ac.uk' and 'man.ac.uk' and the lowest edit distance calculated. A threshold of '3' was used in the following analysis:

Result	(Distance/threshold) Action	Realm
Identical	Route locally	manchester.ac.uk
Identical	Route locally	man.ac.uk
Within threshold	(1<=3): M/S: Reject	manchesterac.uk
Within threshold	(1<=3): M/S: Reject	manchwster.ac.uk
Within threshold	(1<=3): M/S: Reject	manchester.wc.uk
Within threshold	(1<=3): M/S: Reject	manchestet.ac.uk
Within threshold	(1<=3): M/S: Reject	manhester.ac.uk
Within threshold	(1<=3): M/S: Reject	mancheser.ac.uk
Within threshold	(1<=3): M/S: Reject	manchesster.ac.uk
Within threshold	(1<=3): M/S: Reject	mnchester.ac.uk
Within threshold	(1<=3): M/S: Reject	mamchester.ac.uk
Within threshold	(1<=3): M/S: Reject	manxhester.ac.uk
Within threshold	(1<=3): M/S: Reject	manchested.ac.uk
Within threshold	(1<=3): M/S: Reject	manchestter.ac.uk
Within threshold	(1<=3): M/S: Reject	manchestee.ac.uk
Within threshold	(1<=3): M/S: Reject	mancheter.ac.uk
Within threshold	(1<=3): M/S: Reject	mancheater.ac.uk
Within threshold	(1<=3): M/S: Reject	manchrster.ac.uk
Within threshold	(1<=3): M/S: Reject	manchester.au.uk
Within threshold	(1<=3): M/S: Reject	manchedter.ac.uk
Within threshold	(1<=3): M/S: Reject	mancester.ac.uk
Within threshold	(1<=3): M/S: Reject	nanchester.ac.uk
Within threshold	(1<=3): M/S: Reject	manchestera.ac.uk
Within threshold	(1<=3): M/S: Reject	manchester.ac.uj
Within threshold	(1<=3): M/S: Reject	manchedster.ac.uk
Within threshold	(1<=3): M/S: Reject	manchester.act.uk
Within threshold	(1<=3): M/S: Reject	manchester.ac.uk
Within threshold	(1<=3): M/S: Reject	manchester.ax.uk
Within threshold	(2<=3): M/S: Reject	manchster.ac.uk
Within threshold	(2<=3): M/S: Reject	manchestet.ac.ul
Within threshold	(2<=3): M/S: Reject	manchester,ac,uk
Within threshold	(2<=3): M/S: Reject	manchester.co.uk
Within threshold	(2<=3): M/S: Reject	manchester.ac.
Within threshold	(2<=3): M/S: Reject	Manchester.ad.uk
Within threshold	(2<=3): M/S: Reject	mahcester.ac.uk
Within threshold	(2<=3): M/S: Reject	manchester.ca.uk
Within threshold	(2<=3): M/S: Reject	manchedyer.ac.uk
Within threshold	(2<=3): M/S: Reject	machester.ac.uk
Within threshold	(3<=3): M/S: Reject	manchester.ac
Within threshold	(3<=3): M/S: Reject	manchester.org.uk
Within threshold	(3<=3): M/S: Reject	manchester.edu.uk

Within threshold	(3<=3): M/S: Reject	manchester.ac.co.uk
Within threshold	(3<=3): M/S: Reject	ds.man.ac.uk
Within threshold	(3<=3): M/S: Reject	manchester.ac
Exceeded Threshold	(4> 3): forward	manchester.ac.uk.com
Exceeded Threshold	(4> 3): forward	msnchester.ac
Exceeded Threshold	(4> 3): forward	msnchester.ac
Exceeded Threshold	(5> 3): forward	manchester.ac
Exceeded Threshold	(5> 3): forward	manchester.
Exceeded Threshold	(6> 3): forward	yahoo.co.uk
Exceeded Threshold	(6> 3): forward	manchester
Exceeded Threshold	(6> 3): forward	manchester
Exceeded Threshold	(6> 3): forward	student.ac.uk
Exceeded Threshold	(6> 3): forward	manchester
Exceeded Threshold	(6> 3): forward	Manchester.
Exceeded Threshold	(7> 3): forward	gmail.com
Exceeded Threshold	(7> 3): forward	hotmail.co.uk
Exceeded Threshold	(7> 3): forward	yahoo.cn
Exceeded Threshold	(8> 3): forward	yahoo.com
Exceeded Threshold	(8> 3): forward	convidado
Exceeded Threshold	(8> 3): forward	live.com
Exceeded Threshold	(9> 3): forward	student.mancheter.ac.uk
Exceeded Threshold	(9> 3): forward	postgrad.manchester.ac.uk
Exceeded Threshold	(9> 3): forward	manchester.student.co.uk
Exceeded Threshold	(9> 3): forward	student.mancester.ac.uk
Exceeded Threshold	(9> 3): forward	unimail.com
Exceeded Threshold	(9> 3): forward	hotmail.com
Exceeded Threshold	(10> 3): forward	outlook.com
Exceeded Threshold	(12> 3): forward	googlemail.com
Exceeded Threshold	(12> 3): forward	visitor.local
Exceeded Threshold	(13> 3): forward	3gppnetwork.org
Exceeded Threshold	(14> 3): forward	3gppnetworks.org

Appendix 2: Analysis of eduroam realms

All realms compared with all other realms (excludes 0 edit distance matches).

Distance	Count
Distance 1	130
Distance 2	1516
Distance 3	7399
Distance 4	10069
Distance 5	10439
Distance 6	17741
Distance 7	23947
Distance 8	26701
Distance 9	28727
Distance 10	28245
Distance 11	27818
Distance 12	23934
Distance 13	23093
Distance 14	22461
Distance 15	20756

Distance 16	16442
Distance 17	12908
Distance 18	12392
Distance 19	11446
Distance 20	9164
Distance 21	6338
Distance 22	4458
Distance 23	3277
Distance 24	1907
Distance 25	937
Distance 26	488
Distance 27	339
Distance 28	128
Distance 29	18
Distance 30	2

Appendix 3: Analysis of Govroam realms

All realms compared with all other realms (excludes 0 edit distance matches).

Distance	Count
Distance 1	4
Distance 2	17
Distance 3	89
Distance 4	161
Distance 5	169
Distance 6	367
Distance 7	569
Distance 8	744
Distance 9	750
Distance 10	770
Distance 11	797
Distance 12	706
Distance 13	663
Distance 14	516
Distance 15	521
Distance 16	374
Distance 17	218
Distance 18	120
Distance 19	86
Distance 20	87
Distance 21	65
Distance 22	96
Distance 23	77
Distance 24	28
Distance 25	30
Distance 26	26
Distance 27	23
Distance 28	14
Distance 29	22
Distance 30	14

Distance 31	1
Distance 32	4

Appendix 4: Benefits of using gRPC

“gRPC is a modern open source high performance RPC framework that can run in any environment. It can efficiently connect services in and across data centers with pluggable support for load balancing, tracing, health checking and authentication. It is also applicable in last mile of distributed computing to connect devices, mobile applications and browsers to backend services”

gRPC is a client/server based set of libraries which can be used to create applications that needs to run procedures remotely and tranfer information.

Its key advantages here are that it's lightweight (the client needs to be able to start up quickly), low latency (critical for the number of requests to handled), 10 languages supported (cross-platform development required) and well defined.

It has features like built-in timeouts, language-independent data definitions and load balancing. Using this framework saves having to develop something similar, or using heavy-weight approaches such as REST.

This application is very simple in that there is a single type of request (Does the distance between these two strings exceed a threshold?) and a single type of response (yes/no). It just has to do this quickly.

The range of languages makes development easier e.g. initially prototype the client and server in an interpretted language like Python or PHP (which are quick to write but slow to run) and once the structure is clear, rewrite the parts in a compiled language such as C++. The interface between the client and server is language independent so using a python client with a C++ server will work. Similarly developing for Windows platforms can be done with Java or Python and then compiled in C#, for example.