Aura

A peer-to-peer reputation system

Cat Okita

INDEX

Abstract

Current reputation systems are limited in scope, typically in use only on a single site, with no cross-linkage. Aura is a peer-to-peer system designed to provide standalone and integrated trust metrics to individuals and applications. Potential applications include spam tracking, establishment of a decentralized web-of-trust, credibility ratings for websites and mailing lists, and system monitoring.

Introduction

Reputation systems are a means of describing social trust networks. The basic concept behind any given reputation system is a democratic meritocracy. A rating system is used to evaluate individual members, and those results are then collated to produce a consensus about the merit of any given member. The greater the number of data points about any given member, the greater the accuracy of the valuation[1].

Mathematically, there are many different methodologies for computing the merit of the individual. At the most basic level, one could use the arithmetic mean, median and mode as methods of describing the trust level and variance in trust ascribed to a given entity. Adding variables like the reputation rating of the entities contributing to the rating or the time that the rated entity has been known, can increase the perceived accuracy of the derived rating.

In terms of practical applications, a distributed reputation system can be used to share a wide variety of information. For example, rather than relying updates to a centralized blacklist to provide information about spam sources, trusted entities could rapidly exchange new updates as they become known locally.

Socially, information about shared interests, such as good musicians (although not the music itself – that's a different type of software) could be swiftly acquired via entities that have a reputation for similar tastes.

Background

Reputation systems are a means of assigning a quantifiable trust metric to an identifier. Well known examples of this include Livejournal, Friendster, eBay, Advogato and Slashcode-based sites.

---

[1] This theory fails when ratings of a given entity are diametrically opposed, leading to a false neutral.

Livejournal and Friendster are both more accurately described as social networks, rather than reputation systems. However, both sites do provide primitive ranking systems through the ability to describe a person as a friend.

eBay's system of user feedback is one of the best known reputation systems in current use. Buyers and sellers can leave commentary about transactions, and the number of successful transactions, and the amount of feedback (good or bad) strongly affect buyer and seller perceptions. Many sellers will not accept certain types of payment from buyers with low transaction counts – some will not accept bids at all!

Slashcode moderation is a classic example of a reputation system. Postings on a Slashcode driven system are assigned an initial default value [by default 0 for anonymous users, and 1 logged in users]. Users are then assigned 'karma' points, which also start at a default value of zero. Users gain or lose karma when their posts are moderated. Any user with a non-negative karma can become a moderator. Moderators are assigned points which they can use change the value of any post. Status as a moderator is time-limited, to encourage participation.

Advogato is designed to function as an experimental reputation system, and has a three-tiered ranking system. A detailed description is available on their web site, but entities are essentially evaluated by all other entities at or above their level, based on the presentation of peer certificates.

All of these sites have several commonalities:

1. Identification – users must be reliably and securely identifiable
2. Peer interaction – users are evaluated by their peers
3. Feedback – user status changes based on peer interaction

Requirements

The success of any reputation system depends on widespread usage. This implies a number of functional requirements.

- Immediate benefit (or sufficient novelty)
- Easy to use, install and configure
- Lightweight
- Multiplatform
- Secure

Beyond basic functional requirements, a reputation system requires

1. Entity identification, verification and non-repudiation

2. Entity valuation
3. Entity feedback mechanisms

Finally, a reputation system requires significant amounts of data on which to base valuation and feedback.

Methodology

In order to create a broad, accessible user base with few restrictions, it is necessary to eliminate the use of a centralized server. Rather than rely on a single source, such as the classic web model, it is important to create a distributed web of trust.

Classic social networks consist of linked partially-meshed groups. The technological equivalent is peer-to-peer networking, with locally tunable algorithms. Interconnections are based on individual criteria, and information is stored locally.

The software needs to be able to perform two separate tasks – establish trust or credibility, and exchange valuation information. These can be described more succinctly as Reputation and Gossip.

Reputation

Reputation needs to establish the following items about any given entity:

- Identity
- Trust relationship(s)
- Valuation/Credibility

Each entity taking part in a reputation exchange must be uniquely identifiable via public/private key cryptography, associated with an easy to remember identifier such as an email address, FQDN or other similar identifier.

Reputation consists of two ratings, one specific to the user, and the other an aggregate credibility rating composed of information calculated from one (or more) templates.

Gossip

Gossip is the exchange of data which concerns an entity other than the communicating parties. Gossip about an entity must always be identified with the degree of separation from the gossiping entity. Templates are used to structure and constrain gossip and entity-to-entity data exchange.

As is true in human communications, gossip can be both positive and negative, and should always be considered carefully, and evaluated according to the source and content.

Templates are used to describe the format and expected content of a given query. Typical template contents are an item (or series of items), and characteristics of that item. Templates are structured to require numeric responses to set queries, for performance and storage reasons. The returned template data can then be used to compute a template specific reputation, and contribute towards the overall reputation of an entity.

A trivial example of a template fragment is shown below.

```
TemplateName=Stable_Mail_Server

Server_Name?
        Sendmail:0
        Qmail:1
        Postfix:2
Stable?
        Yes:1
        No:0
```

Figure 1 - Template Fragment

Using the template above as an example, the data transmitted would be the name of the template (Stable_Mail_Server), the signature of the template, and presuming a stable Sendmail server, the values 0 and 1.

From this example, we could calculate that Sendmail has a reputation of being a stable mail server. The actual reputation calculation process involves more variables, and variable weighting.

Design Issues

The specific design issues under consideration in the process of implementing Aura are as follows:

- Secure
    - Data exchange must be encrypted.
    - Data specific to a given entity must be keyed only to that entity.

    - Entity identifiers must be unique [public/private key], and may not be readily repudiated.
    - Templates must be signed
    - Information poisoning must be non-trivial
    - Key storage should be encrypted and permission limited
- Fast
- Efficient
    - No significant system or network load should be produced under standard usage (up to 100k entity records estimated per individual)
    - Multiple users on any single server may share the same server and data store, although user specific data will always be keyed to that user.
- Simple
    - Straightforward to install and configure
    - Human readable data where possible
    - Templates should be text-based[2]
- Portable
    - Should have limited dependencies on OS or hardware specific items
    - Entity keys should be transferable between platforms or via devices such as USB key drives
- Flexible
    - User extensions should be trivial
    - Database integration should be trivial
- UI
    - Command line for scripting and power users
    - GUI for standard use [planned]

Design

Aura is implemented as a separate client and server process, with the server handling all incoming queries for information, and the client handling end user interaction. This is an artificial distinction to allow database queries and comparisons to be separated from interface design considerations.

Functionally, Aura is designed to function both as a standalone application, and as a support for plug-ins to a variety of applications, such as mail clients or monitoring agents.

As a part of the initial discovery process, aura will attempt to contact either the default seed server(s), or a user specified set of addresses to establish a list of

---

[2] ASCII or Unicode as appropriate

aura speakers. Subsequent invocations will default to attempting to contact speakers in aura's immediate web of trust.

In situations where the contact mechanism for an entity is unknown, Aura implements path-based message passing. This implies that any entity may request any other entity to act as a message passing mechanism, recording the route to contact, and passing a message to a difficult to reach or unknown entity. The establishment of direct contact between entities is the preferred mechanism, as it absolves messenger hosts of the burden of additional traffic.

It is expected, but not required that the Aura daemon (aurad) will run in the background, and maintain a limited set of 'live' connections to entities in its trust list.

The initial connection between two previously unknown entities consists of an exchange of public keys and the associated identifier[3]. Unless the user intervenes, newly discovered entities are assigned a default reputation value. From this point onwards, all information sent between entities will be encrypted with the private key of each entity, to establish non-repudiation. Next, the signatures of the templates used by each entity are exchanged. Subsequently, the two entities exchange template-based information about themselves, and gossip about other entities.

Connections between known entities consist of updates to template-based information, further exchange of gossip, and message passing.

The type and amount of information exchanged can be limited according to reputation, template type, and entity. Gossip must always contain the degree of indirection (number of hops) from the originator.

Reputation changes are made based on a combination of user input, gossip from trusted entities, trust equations and time. While volume will ultimately damp sudden changes in reputation based on gossip, initial swings are to be expected, while the system settles into equilibrium.

Information poisoning is one of the greatest challenges in implementing a successful reputation system. Aura is potentially vulnerable to several information poisoning attacks. Template poisoning (where a trusted source for templates is successfully compromised, and templates replaced), identity theft or

---

[3] An entity is not considered to be 'known' for this purpose until a key exchange has taken place. Entities which have been gossiped about indirectly will have an identifier, but not a public key associated with them.

forging, and flooding are all possible vectors. Like any software, Aura is also vulnerable to code alteration or replacement.

The effect of most attacks is strongly affected by the total number of entities involved in the aura network. As with reputation values, a smaller system will show changes at a greater rate.

Template poisoning is mitigated by requiring that all templates be signed, and that signatures match prior to any exchange of information.

Identity theft and forging are addressed through the use of public/private key cryptography, requiring that any attacker either perform a successful decryption, or obtain key phrases through other mechanisms.

It is possible to forge an identify from the initial contact onwards, and thus it is recommended that trust be considered carefully, and validated where possible. The GPG Privacy Guide provides an excellent example of entity verification procedures[4].

Flooding is the most challenging attack to defeat, but while sheer volume of queries could cause Denial of Service attacks (limited by the number of connections available), a lack of trust, as described in Advogato's trust metric1 would inhibit information poisoning as a result.

Implementation

Aura is implemented in C, using Peter Gutman's Cryptlib, and SQLite as a database backend.

The basic structure consists of a simple TCP daemon, which listens for incoming queries, identifies and processes them appropriately, when contacted by a client.

All data is stored locally in one database file per locally known entity[5]. This has the effect of making it easy to lock data to each local user.

When the aura client is first started, it will check to see if it is running for the first time, and prompt for the location or creation of a public/private key pair. Once a key has been established, the client contacts the local server to log in.

---

[4] http://www.gnupg.org/gph/en/manual.html#AEN554

[5] This is a result of using SQLite, the desire to keep user data separated, portable and secure.

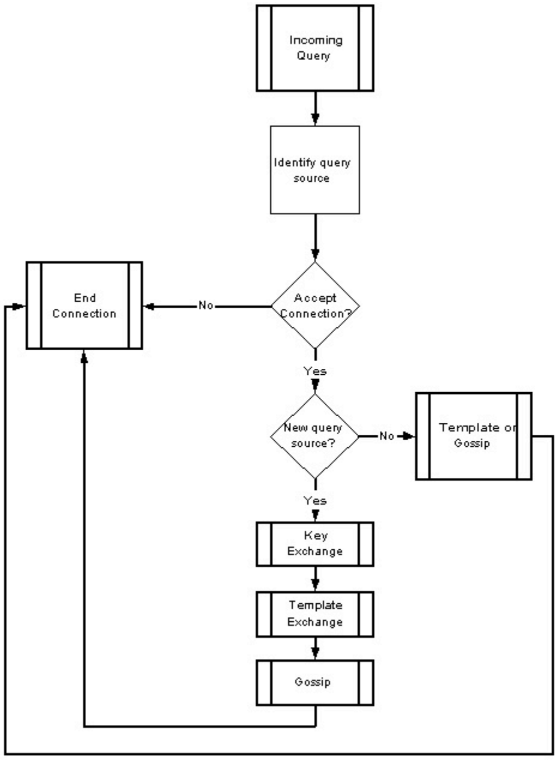
The server will then create a database file for the entity, and add the entity to the list of entities for which it maintains canonical information.

When a server is first started, it will attempt to contact other aura speakers, either via a local list, or by contacting a remote seed server.

If a server is designated as a seed server, it will receive queries from aura clients, seeking other aura speakers. The seed server expects to receive an ID string from the client, containing the IP and version of the aura client. The seed server then returns a list of IPs. The seed server may also return a notification message, such as failure notice, or the availability of a new version.

A message passed to a server to contact a local entity will be handled as a standard query. If the message passed is for an unknown entity, the server will randomly select one of its established connections, check to ensure that a routing loop is not being created, and forward the message to the selected aura speaker.

An initial client request from a previously unknown client contains the entity identifier of the entity they are attempting to contact (typically either an email address or FQDN), their identifier, and the associated public key. The server will return its entity identifier[6] and public key encrypted with the public key of the client. Subsequent exchanges establish common templates, exchange data, and exchange gossip.



A client can either function locally, querying the server for known information, or remotely, requesting new information and updates. Locally, a basic query about an entity identifier will return the private and public reputation of the queried entity. The data key is of the format <entity identifier><key hash>, returning <entity identifier><private reputation><public reputation>. Additional information requests or changes can be specified via flags.

Each entity entry contains a hop-

---

[6] In the event that the server handles information for multiple entities, the appropriate entity is selected.

limited list of the entities that have gossiped about it. While it is possible to maintain a list of all entities, no matter how distant that have gossiped about a given entity, it is definitely impractical!

Configuration

The configuration files are text based and can be set individually, and for the entire server. It is possible to configure levels of trust, and what information will be shared with whom, based on reputation (public/private), specific entity identifier and template.

Critique

It is not possible to determine the actual success of this software without a significantly sized and active population.

Related work

There are a large number of theoretical papers available at the Reputations Research Network, at the University of Michigan. More practically, Advogato , Ralph Levien's web module focusing on group trust models for peer certification, eBay and SpamAssassin are practical examples of limited focus reputation systems.

Conclusions

Further testing in the wild, or in a large scale test environment is indicated.

Availability

Aura is released under the Berkeley Database License (better known as the Sleepycat license), and is fully GPL compatible. It can be downloaded from:

http://www.geekness.net/tools/aura

The most current version of this document may also be found at the above location.

Acknowledgements

The seeds of this software are derived from several sources – a strong interest in AI and swarm intelligence – Caezar's Challenge at Defcon X, discussing biological models in computation – Neal Stephenson's fictional meme propagation – and a number of years collecting statistics for backbone networks.

Thanks for patience, inspiration and/or editing go to Mark Langston, Mikael Olsson, Matthew Ringel, Kim Wallace, Dan Foster, Agent X and Myles.

References

1. Advogato's Trust Metric - http://www.advogato.org/trust-metric.html
2. http://www.fsf.org/licenses/license-list.html
3. Livejournal – http://www.livejournal.com
4. Friendster – http://www.friendster.com
5. Slashcode – http://slashcode.com and http://sourceforge.net/projects/slashcode/
6. Slashcode moderation - http://sagewire.sage.org/moderation.shtml
7. eBay – http://www.ebay.com
8. SpamAssassin – http://www.spamassassin.org
9. GnuPG – http://www.gnupg.org