Adding Fake Facts to Ontologies

Fabian M. Suchanek
Max Planck Institute for Informatics, Germany
& INRIA Saclay, France
fabian@suchanek.name

David Gross-Amblard
IRISA, Université de Rennes 1, France
& INRIA Saclay, France
david.gross-amblard@univ-rennes1.fr

ABSTRACT

In this paper, we study how artificial facts can be added to an RDFS ontology. Artificial facts are an easy way of proving the ownership of an ontology: If another ontology contains the artificial fact, it has probably been taken from the original ontology. We show how the ownership of an ontology can be established with provably tight probability bounds, even if only parts of the ontology are being re-used. We explain how artificial facts can be generated in an inconspicuous and minimally disruptive way. Our demo allows users to generate artificial facts and to guess which facts were generated.

1. INTRODUCTION

Public Ontologies. An ontology is a formal collection of world knowledge. Ontologies on the Semantic Web are usually available under some kind of license, such as the Creative-Commons License (CC)[1] or the GNU General Public License (GPL)[2]. Table 1 lists some of the major ontologies with their licenses [5].

License	Conditions	Ontologies
GPL	attribution, copyleft	SUMO [4]
CC-BY-SA	attribution, share-alike	DBpedia [2]
CC-BY	attribution	YAGO [6], Freebase, Geonames, OpenCyc [3]
CC-BY-ND	attribution, no derivatives	UniProt
proprietary	access under restrictions	TrueKnowledge, full Cyc [3]

Table 1: Common licenses for ontologies

Most licenses give users a free hand to use the data as they please. The user may even re-publish the ontology as part of another ontology. In this case, the user has to give credit to the original ontology. Commercially sold ontologies (such as Cyc) usually have more restrictive licenses. They prohibit re-publication completely. Some ontologies, such as TrueKnowledge[3], have even more restrictive licenses. They prohibit even the systematic retrieval of the data. In all of these cases, the systematic dissemination of the data without proper acknowledgment to the original is prohibited.

Ownership Proof. This raises the question of how we can detect whether ontological data has been illegally disseminated. We call a person who re-publishes an ontology (or part of it) in a way that is inconsistent with its license an *attacker*. We call the source ontology the *original ontology* and the re-published ontology the *stolen ontology*. The attacker could, e.g., re-publish the original ontology under his own name. Or he could use parts of the original in his own ontology without giving due credit to the original. We call an ontology that is potentially stolen a *suspect ontology*. In this paper, we do not deal with the ethical or legal consequences of re-using data from an ontology. The reuse may be prohibited or entirely legal. In this paper, we just establish whether a reuse took place or not. That is, we deal with the general question of *ownership proof*: How can we attest that the suspect ontology contains part of the original ontology?

Obviously, it is not sufficient to find that the suspect ontology contains data from the original ontology. This is because ontologies contain world knowledge, which anybody can collect. The attacker can simply claim that he collected the data manually, or that he extracted the statements from public sources, and that he happened to produce the same data as the original ontology. This might even be true. Alternatively, we could publish the original ontology with a time stamp (in the style of proof of software ownership). For example, we could upload the ontology to a trusted external server. If someone else publishes the same data later, then we could point to the original copy. However, this does not prove our ownership. The other publisher could have had the data before we published ours. The fact that he did not publish his data cannot be used against him.

Related Work. Previous work (e.g., [1, 5]) has focused on sophisticated approaches for ownership proofs, notably on *watermarking*. Watermarking introduces small modifications into the original ontology. If these modifications appear in the suspect ontology, they act as a proof of ownership. Our own work [5] has introduced the concept of *subtractive watermarking*, where the ontology is marked by removing certain statements. All of these approaches have a considerable drawback: To prove that a suspect ontology

[1] `http://creativecommons.org`
[2] `http://www.gnu.org/copyleft/gpl.html`
[3] `http://www.trueknowledge.com`

WWW 2012, Lyon/France

"stole data" from the original ontology, the suspect ontology has to be downloaded and checked for the marks. Since the watermarking hides only one bit of information per mark, there are usually several dozens to several hundred marks that have to be checked in order to have reliable indication of theft. Checking these marks requires downloading the ontology, unpacking it on the local hard drive, and running an algorithm that is at least linear. This is a very time-consuming process. It might even be impossible to check the marks, if the suspect ontology cannot be downloaded, but only queried online. There is currently no way an ontology owner can check manually online whether some ontology is stolen.

Contribution. This is where the current paper steps in. We propose to use *additive watermarking*. This technique is inspired by the use of fictitious entities in dictionaries or *trap streets* in maps that serve to identify plagiarism. In this paper, we transfer this idea to ontologies. Our technique adds a small number of artificial facts to the original ontology. If these artificial facts appear in a suspect ontology, they act as a proof of ownership. The appearance of an artificial fact in a suspect ontology can be checked manually in no time. It suffices to query the suspect ontology for the artificial fact and see whether it appears. In this paper, we substantiate this idea by three contributions:

1. We give an algorithm that generates artificial facts automatically in an inconspicuous way.
2. We give a theoretical analysis that yields the optimal number of facts that have to be added to establish ownership with provable bounds.
3. We provide a demo system that lets users add artificial facts and play with them.

2. PRELIMINARIES

Watermarking. A *watermarking protocol* is defined by two algorithms, a *marker* and a *detector*. Given an original ontology and a secret key, the marker generates a watermarked ontology. Given a suspect ontology, the original ontology, and the secret key, the detector decides if the suspect ontology contains the mark. If that is the case, it is assumed that the suspect ontology stole data from the original ontology. It may be, however, that the suspect ontology is innocent and that it contains the mark just by chance. We call such cases *false positives*. The marking algorithm is designed in such a way that the probability of a false positive is provably below a security threshold ξ. For common applications, $\xi = 10^{-6}$. The attacker can also attempt to obscure the mark. In this case, the detector cannot detect that the suspect ontology is stolen. The marking algorithm bounds the probability of such cases to be below ξ, too.

Additive Watermarking. Our watermarking works by adding a small number of artificial facts to the ontology. Since this creates factually wrong statements, this technique compromises the quality of the ontology. If the ontology is highly sensitive to even small misrepresentations of reality, then the consequences might be harmful. Therefore, an ontology owner has to decide carefully whether watermarking is an option for him or not. As always, watermarking is a trade-off between the ability to prove ownership and the truthfulness of the data.

We show that the total number of fake facts that our algorithm adds is usually small (less than 20 in the experiments). For large ontologies, this number fades in comparison with the total number of facts, which may be in the millions. For automatically constructed ontologies, the number of fake facts is usually orders of magnitudes smaller than the number of false facts that the ontologies contain anyway. YAGO[6], e.g., one of the largest ontologies with a quality guarantee, has an accuracy of 95%, meaning that it contains already hundreds of thousands of accidentally false statements.

Visibility. Our technique aims to generate artificial facts that cannot be easily spotted by a human or detected by a machine. For this purpose, the facts re-use an existing relation from the ontology, an existing entity as subject and either another existing entity or a statistically invisible literal value as object. For example, our method could add the fact that someone's height was 1.76 meters. This value is chosen to be both valid for an ownership proof and statistically plausible. For the statistical plausibility, we rely on a *statistical invisibility threshold* θ, with $0 < \theta < 1$. This threshold is fixed upfront by the user. If θ is close to 0, then the artificial facts will be less visible, but the algorithm will also have fewer opportunities for marking. If θ is close to 1, then the algorithm can mark more entities, but the marks are potentially more visible.

CSPRNGs. Our watermarking makes use of a cryptographically secure pseudo-random number generator (CSRNG). A CSPRNG is a function that, given an integer seed value, generates a pseudo-random sequence of bits. A CSPRNG is designed in such a way that it is close to impossible (1) to predict the next bit without knowing the seed value or (2) to find the seed value that generated the bit sequence. We use a fixed CSPRNG $\mathcal{G}$. By combining several random bits, $\mathcal{G}$ can also produce a pseudo-random integer value. We write $\mathcal{G}.nextInt(n)$ to denote the next pseudo-random integer value of $\mathcal{G}$ between 0 and $n-1$ inclusively.

RDFS. Our approach targets RDFS ontologies [7]. An RDFS ontology over a set of entities E, a set of literals L, and a set of relation names R can be seen as a set of triples $O \subset E \times R \times (E \cup L)$. Each triple is called a *statement*, with its components being called *subject*, *predicate* and *object*. RDFS can also designate entities as *classes*, and other entities as *instances* of one or multiple classes. We assume that every entity is either a class or an instance of a class.[4] A class can be the *super-class* of another class. In this case, all instances of the class are implicitly instances of the super-class. A *direct instance* of a class is an instance that is not implied by a super-class relationship. In all of the following, we see a class as the set of its direct instances. In general, our technique targets RDFS ontologies with rich instances and many assertions about instances. Our approach is less well-suited for ontologies that contain only a schema. Our main target are ontologies such as the ones in Table 1.

Namespaces. An attacker can rename entities and relations in the stolen ontology. He can change the namespace of the data, or apply syntactic variations to names and literals. For our approach, we assume that it is still possible for a human to determine whether a given fake fact appears in the suspect ontology. This assumption is reasonable, because if it were not possible for a human to say whether a given ontology contains a particular fact, then the ontology would be of very limited use.

[4]In RDFS, classes can be instances of other classes, but this is rare in practice.

3. ADDITIVE WATERMARKING

Possible Values. Let us discuss the statements that our watermarking algorithm will add to an ontology O. The algorithm will add different types of statements for different types of relations. Let us first consider a relation r whose objects are entities or non-numeric literals. A *possible value* for r in a class c is an entity (or literal) v such that

$$|\{e \in c : \langle e, r, v\rangle \ \in O\}| > 1/\theta$$

(where θ is the statistical invisibility threshold). This means that if we add an edge $\langle e, r, v\rangle$ to O, the number of subjects of r that have v as an object will change by less than $1/\theta$. For example, with $\theta = 0.01$, we can add a Grammy Award winner to the ontology, but not a Fields Medal winner, because there are less than 100 Fields Medal winners.

Let us now look at a relation r whose objects are rational numbers or integer numbers. The *image* of r in a class c is

$$im(r, c) = \{v | \exists e \in c : \langle e, r, v\rangle \in O\}$$

A *possible value* for r in c is any value

$$v \in [min(im(r, c), max(im(r, c))]$$

that has at most as many decimal mantissa digits as the values in $im(r, c)$ do. If the range of r is dates, then a *possible value* for r is any date *yyyy-mm-dd*, where *yyyy* is a possible value of r projected to the years and $mm \in [1, 12]$ and $dd \in [1, 28]$.

Markable Entities. The *markable relations* of an entity e are all relations that have possible values in any class of which e is a direct instance. The *defining class* of a markable relation r for an entity e is the class c such that (1) e is a direct instance of c, (2) r has possible values in c, and (3) the set of possible values is smallest among all the classes of e in which r has possible values. We note the defining class of r for e as $def(r, e)$. Then, the *possible values* of r for e are simply the possible values of r in the defining class. The *maximal cardinality* of r for e is the maximal number of outgoing r-links that members of the defining class exhibit:

$$maxcard(r, e) = max_{e' \in def(r,e)} |\{e'' : \langle e', r, e''\rangle \in O\}|$$

These definitions are just one possible way to designate possible values. Indeed, the value invention mechanism can be adapted to the ontology, since our marking algorithm treats it as a black box.

Marking. Algorithm 1 shows how we watermark an ontology. The ontology owner first chooses a secret key, i.e., a numeric value that only he knows. He also chooses the number of facts n that he wishes to generate (we will discuss in Section 4 how to choose n). The CSPRNG $\mathcal{G}$ is seeded with the hash code of the ontology and the secret key. Then, our algorithm iterates through all N entities of the original ontology[5]. For each entity e, $\mathcal{G}$ is asked to generate a random integer number between 0 and N/n. If this number equals 0, e is chosen to become part of an artificial fact. The algorithm uses $\mathcal{G}$ again to choose a markable relation r_i of e, and then again to choose a value v_j among the possible values. If $\langle e, r_i, v_j\rangle$ already exists in the ontology, e is skipped as unmarkable. If e already has the maximum number of r links, then one r-fact is deleted at random from e. Then the artificial fact $\langle e, r_i, v_j\rangle$ is added to the ontology. This process is repeated until n facts have been added.

[5]We assume a canonical ordering, e.g., alphabetical.

Algorithm 1 additiveMark(ontology O, key $\mathcal{K}$, int n)

```
N ← number of entities in O
G.seed(hash(O) ⊕ K)
while true do
  for all entities e of O do
    if G.nextInt(N/n) ≠ 0 then continue
    r_1, ..., r_k ← markable relations of e
    if k = 0 then continue
    i ← G.nextInt(k) + 1
    v_1, ..., v_l ← possible values of r_i for e
    j ← G.nextInt(l) + 1
    if ⟨e, r_i, v_j⟩ ∈ O then continue
    if |{v : ⟨e, r_i, v⟩ ∈ O}| = maxcard(r, e) then
      remove one statement ⟨e, r_i, v⟩ at random from O
    end if
    O ← O + ⟨e, r_i, v_j⟩
    if n facts have been added then return
  end for
end while
```

Detection. Our detection algorithm simply checks if the added n facts are present in the suspect ontology. This can even be done manually. To prove the ownership, the ontology owner shows the secret key that generated the artificial facts. By the nature of $\mathcal{G}$, it is impossible that the ontology owner created a posteriori the secret key that generates the artificial facts. Therefore, the ontology owner must have generated the facts by himself based on the key. This proves his ownership.

4. ANALYSIS

Consistency. We have to ensure that the marked ontology is still logically consistent. Our approach considers RDFS ontologies only. RDFS contains no class disjointness constraints, no domain/range consistency constraints (the class membership is simply inferred), and no cardinality constraints. All deduction rules in RDFS are purely positive. Therefore, by design, an RDFS ontology cannot become inconsistent. Therefore, we may add (and even remove) assertions about instances without the danger of inconsistency.

False Positives. Let us now model the probability that a random ontology is considered stolen by mistake. For this purpose, we have to estimate the probability that an artificial fact appears in an innocent ontology by chance. We say that the *expected probability* of a possible value v for a relation r in a class c is the proportion of entities in c that have this value for r:

$$p(r, v, c) = \frac{|\{e \in c : \langle e, r, v\rangle \in O\}|}{|c|}$$

If $p(r, v, c) = 0$, we set $p(r, v, c)$ to one over the number of possible values of r in c. The probability that a generated fact $\langle e, r, v\rangle$ in a class c appears by chance in an innocent ontology is far smaller that $p(r, v, c)$, because the original ontology represents only part of reality, and many more possible values may be admissible in reality. Let p be the maximal expected probability among the generated values. Since our algorithm chooses the artificial facts independently, the probability that all n facts appear by chance in any given innocent ontology is bounded by p^n. Therefore, choosing $n \geq \frac{\ln \xi}{\ln p}$ ensures that the probability of a false positive is below the security threshold ξ. In reality, Algorithm 1 can

simply be run until the product of the expected probabilities of the added facts is below the security threshold ξ.

Subset Attacks. Let us consider an attacker who steals only a part of the ontology. Let us assume that the attacker steals random facts with a probability q. That is, if the original ontology contains k facts, the stolen portion will have an expected size of $q \cdot k$. To guarantee detection, we have to ensure that every sub-portion of size $q \cdot k$ contains at least n artificial facts. A Hoeffding bound tells us that we have to add N facts in total, where N is such that

$$(q - n/N)^2 N \geq -ln(\xi)/2$$

This choice ensures that the probability that a sub-portion contains less than n facts is less than ξ. This allows the ontology owner to compute the number of artificial facts to add, if he wishes to protect the ontology against thefts of a certain size.

Experiments. We were interested in the number of facts that we have to add to existing ontologies in order to protect them against theft. For this purpose, we collected 5 ontologies from the Semantic Web that cover a wide range of complexity, size, and topics (Table 2): The core part of YAGO [6], the manually supervised part of DBpedia [2], the Universal Protein Resource[6], an ontology about schools (provided by the UK government[7]), and a subset of the IMDb[8]. Table 2 shows the ontologies with the number of facts, the number of relations, and the number of instances. We report the proportion of markable instances, the proportion of relations that have possible values and the average number of possible values per entity. We determine the number of facts to add by help of the average expected probabilities. Only a handful of facts have to be added to protect the ontologies against theft. Even if we want to guard against subset attacks ($q = 80\%, q = 50\%$), the number of added facts is minuscule in comparison with the huge number of facts in the ontologies. The loss in precision is hardly noticeable.

	YAGO	DBpedia	UniProt	Schools	IMDb
# facts	18m	20m	6096	6m	34m
# relations	83	1107	4	186	12
# instances	2.6m	1.7m	1869	263,954	4.6m
markable inst.	45%	99%	71%	50%	100%
markable rel.	81%	50%	25%	53%	75%
avg.# poss. val.	2.6m	158m	60	6k	6k
Facts to add	1	1	4	2	2
- for $q = 80\%$	14	14	20	16	16
- for $q = 50\%$	32	32	43	36	36

Table 2: Marking different ontologies

Visibility. Our approach generates facts only on the basis of class membership. It does not take into account other facts of the marked entity. This is a weakness. Our approach could, e.g., generate a death date that is earlier than the birth date. Or it could generate a surface area for a city that is implausible given its population. Although our approach will never generate facts that make the ontology inconsistent in the logical sense, it could generate facts that are semantically implausible. However, even though this will certainly happen in practice, our experience with real ontologies makes us believe that the majority of artificial facts will go unnoticed. Our goal is to test this hypothesis empirically with the present demo.

[6] http://www.uniprot.org/
[7] http://data.gov.uk/
[8] http://imdb.com

5. DEMO

Our demo visualizes the creation of fake facts for ontologies. The user chooses an ontology and our system displays the ontology in a graph-like fashion. Then the user can instruct the system to add fake facts to the ontology. To increase the educational and entertainment value of the demo, the user can explicitly choose the entity to which the fake fact shall be added. Users can also choose the relation for which a fake fact shall be generated, if they wish to. This allows, e.g., adding a long-deserved award to their favorite pop singer, generating a death date for a politician, or adding new actors to a movie. The actual values are chosen by the system in order to be least suspect. The artificial facts appear highlighted in the graph of the ontology. To test the plausibility of the invented facts, users are invited to hide the highlighting of the generated facts and to have other users guess which facts were added to the entity (Figure 1). Since, in a real application, such facts would be chosen and generated automatically, our demo also displays the expected probability of each generated fact, which mirrors the security contributed by this fact.

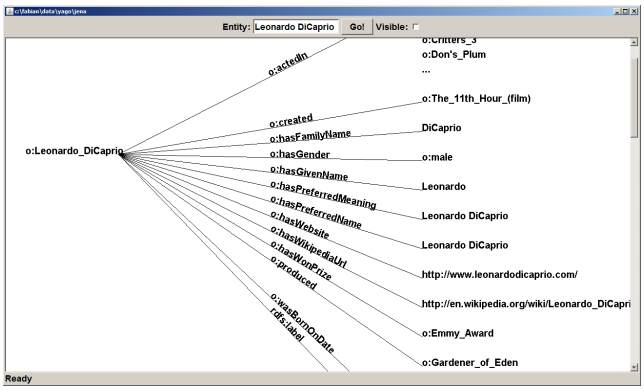

Figure 1: The system added a hidden fake fact to Leonardo DiCaprio on YAGO

6. REFERENCES

[1] R. Agrawal, P. J. Haas, and J. Kiernan. Watermarking Relational Data: Framework, Algorithms and Analysis. *VLDB J.*, 12(2):157–169, 2003.

[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. DBpedia: A Nucleus for a Web of Open Data. In *ISWC*, 2007.

[3] C. Matuszek, J. Cabral, M. Witbrock, and J. Deoliveira. An introduction to the syntax and content of cyc. In *AAAI Spring Symposium*, 2006.

[4] I. Niles and A. Pease. Towards a standard upper ontology. In *FOIS*, 2001.

[5] F. M. Suchanek, D. Gross-Amblard, and S. Abiteboul. Watermarking for ontologies. In *ISWC*, 2011.

[6] F. M. Suchanek, G. Kasneci, and G. Weikum. YAGO: A core of semantic knowledge. In *WWW*, 2007.

[7] Word Wide Web Consortium. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation 2004-02-10.