-----Original Message-----
From: owner-stix-discussion-list@lists.mitre.org [mailto:owner-stix-discussion-list@lists.mitre.org] On Behalf Of Barnum, Sean D.
Sent: Wednesday, February 05, 2014 12:40 PM
To: Wunder, John A.; Terry MacDonald; Kyle Maxwell
Cc: Dave Dittrich; stix-discussion-list Structured Threat Information Expression/ST
Subject: Re: STIX Content Revision and Revocation

Hi everyone.

Okay, so the sickness has wore off and meds have abated. I took some time to look through the discussion here, attempt to understand feedback/input, attempt to extract out some consensus (or leanings that way) and try to model some things to get a feel if there might be any issues/questions with the consensus leanings.

So, out of all of that let me take a swing at my thoughts.

Let me start by reiterating that for right now, our goal must be limited to what we can/should do in STIX 1.1 to provide at least a minimal capability for effective revision & revocation support.
That does not mean that we should not be thinking grander and longer term thoughts. We absolutely should. But we need to remember that our ability to act currently is constrained by our level of understanding of the issue, our confidence in the appropriateness of any consensus solution and pesky things like a need for backward compatibility.
That being said, I think there were two specific topics discussed that are important things to consider for 2.0 but are likely out of scope for 1.1. Those are the intriguing Temporal Graph approach suggested by Pat and the issue of full vs. partial updating of content. We simply do not have time to absorb and consider Temporal Graphs right now and there have been some concerns with complexities/consistencies of specifying and implementing partial updating capabilities (lots of open questions that we cannot fully resolve at this time). We encourage you all to continue to discuss these topics on an ongoing basis but they are likely out of scope for 1.1.

Out of the discussions that led to the initiation of this thread and the discussion that has occurred in the thread thus far, I see the following use cases identified as relevant:

* · Simple create and update of content by a single entity (local evolution)
* · Share an original object
* · Share an updated object (with ability to understand its history for context, deduping, trust decisions, etc.)
* · Share an object with its evolutionary chain (all relevant versions that led from the original object to its current version) (gives local visibility into chain)
* · Share an object with its evolutionary cluster (all evolutionary chains that derived from the object's origin object within a community scope) (gives local visibility into cluster)(this is the full set of a community's knowledge of something the community thinks of as the same thing)
* · Reference a specific version of an object
* · Reference the original version of an object
* · Reference the latest version of an object
* · Reference an evolutionary chain
* · Reference an evolutionary cluster

So, out of the discussion I saw 6 possible options for approaches:

* The original Option #1 from the initial post of this thread (simple ordinal version numbers).
    * Nobody spoke up for this approach and I think we can discount it for now. Please speak up if you disagree.
* The original Option #2 from the initial post of this thread (new versions get new ids and version GUIDs; references can reference just the ID or the ID/Version-GUID pair)
    * Nobody really spoke up for this approach as it was originally described. I think we can discount it for now. Please speak up if you disagree.
* The original Option #3 from the initial post of this thread (new versions get new ids (no version GUIDs))
    * Bernd did not like this approach because he feels it is too complicated for the simple case of internal/local evolution of content.
    * Others agreed that it would be complicated for the simple case of internal/local evolution of content but was an option for a basic solution and had the advantage of being conservative and did not lock us into anything that may prove detrimental to a long term solution.
* Temporal Graphs
    * Seems intriguing and shows promise but is out of scope for 1.1 due to novelty and time constraints.
* Git model
    * Appears intriguing due to its established reputation in distributed code version control
    * Many players expressed concerns/questions over its applicability to version tracking/referencing (as opposed to version control) of STIX content both local and in distributed sharing communities. While I would not rule it out as a potential long term option at this point, I personally share many of these concerns and more.
    * I think it would be good for us all to talk through all these issues but do not think we have the luxury of time or consensus to do that for 1.1. Particularly, the "patch"-centric perspective of the git approach would be out of scope for 1.1 as it relies on partial updates.
    * Kyle, if you want to continue to fight for this approach in 1.1 please let us know.
* Bernd's suggested Option #2A that evolved a good deal over the discussion
    * This approach with its evolution during the discussion seems to be the closest thing to a consensus that I saw.
    * I will try to discuss thoughts on this approach in the remainder of this post.

Option #6 (Modified Option #2A)

(I attempt here to boil down what I read in the thread as well as other context from previous community discussions and some modeling of scenarios)

* Fundamental understanding of the suggested practice that all content shared externally is given IDs within the sharer's namespace
    * If content is being reshared then the shared content (with a sharer namespaced ID) is also given an explicit relationship to the original content and ideally an assertion of the sharer's confidence in the content
* Each version of each item gets an ID & a timestamp
* References to items can be done either with just @idref or with @idref & @timestamp_ref
* Internal revisions can either be implicit or explicit
    * keep same ID and just update the timestamp (implicit) (this supports Bernd's simple common case)

    * include relationship to base content id & timestamp (explicit)
* Revisions based on external content update timestamp and include revision relationship to base content id & timestamp
*
    History construct (should it be included or not?)
    * Answering “Where did item X come from?” – What is its evolutionary chain?
    * With History construct it is explicit and atomic (part of the item itself).
    * Without History construct it must be derived from a blend of explicit relationships (external derivations) and implicit relationships (timestamps) and is not atomic.
* Capabilities of this option
    * Can support version tracking for local/internal evolution
    * Can support sharing of an original identifiable object
    * Can support sharing of an updated identifiable object
        * limited trackability of historical context of internal evolution if implicit approach is used without History construct (historical context is not atomic and must be derived)
        * full traceability of historical context if explicit approach is used without History construct (historical context is not atomic and must be derived)
        * full traceability of historical context if used with History construct (historical context is atomic with object and requires no derivation)
    * Can reference specific version (e.g. @idref=Foo @timestamp_ref=Time:2)
    * Can reference localized evolutionary chain (e.g. @idref=Foo)
        * Does this mean the whole chain, the origin node or the latest node? Opinions differ
    * Can not reference global evolutionary chain
    * Can not reference evolutionary cluster.
* The attached document includes a number of diagrams that I used to think through many of these questions/issues. I think they may be useful for everyone else too.

Some open questions:
* Are we concerned with the increased potential for race conditions on using timestamps rather than GUIDs?
* How are we going to deal with clock alignment issues that come into play whenever timestamps are used for identification/correlation within a collaborative environment?
* How difficult will this approach be for producers of content?
* How difficult will this approach be for consumers of content?
* How are namespaces handled? Does anyone have issues with the suggested practice that all content shared externally is given IDs within the sharer's namespace and that if content is being reshared then the shared content (with a sharer namespaced ID) is also given an explicit relationship to the original content and ideally an assertion of the sharer's confidence in the content?
* How to interpret and manage validity of relationships to objects as they are updated? Are both perspectives outlined in the attached document valid? Do you agree or disagree that it is necessary to support both?
* How to specify if a relationship is intended to be to the latest version in an evolutionary chain, the original version in the chain or to the entire chain?
* What should the vocabulary be for revision relationships?
* Should we add a new @timestamp_ref attribute to all id'd constructs to support referencing timestamps? (I believe this would be necessary for Option #6)

* What field to use for the version timestamp?
    * A Produced_Time field already exists within InformationSourceType already attached to id'd constructs. It is several levels down rather than an attribute on the core construct like @id/@idref are.
        * Should an additional @timestamp attribute be added to each construct?
            * How do we handle this duplication/redundancy with the existing Produced_Time field?
        * Should we simply capture the produced timestamp in the existing field and align/match @timestamp_ref to it?
        * Should we change the format of the existing timestamps to the format Pat suggested (RFC3339 with at last 6 digits of precision for 'time-secfrac')?
            * Should all STIX timestamp formats be changed this way? If not, how do we handle the inconsistencies?

Summary
So, I think there has been some great discussion and some great suggestions brought up in this thread. In particular, Bernd's suggestion for using timestamps instead of version GUIDs seems to have significant potential.
After some careful, though by no means complete and comprehensive, thought it looks to me like Option #6 may make sense for now.
If we do decide to go this route though, we will need to address the open questions above.
The most significant challenges I see with Option #6 are: the inability to support some evolutionary chain/cluster concepts, inherent issues with clocks, and how to handle the issue of where to put the version timestamp info in the schema while maintaining backward compatibility and clarity/consistency.
It looks like the @timestamp_ref capability should be able to be added without affecting backward compatibility.

Please offer your thoughts on the open questions we must consider quickly.
Please offer thoughts on anything we have overlooked or I have completely wiffed on.

And finally, please offer your opinion on whether you think Option #6 as it is outlined above makes sense for pursuing in STIX 1.1.

Thanks everyone,

Sean

From: <Wunder>, John Wunder <jwunder@mitre.org>
Date: Monday, February 3, 2014 9:11 AM
To: Terry MacDonald <terry.macdonald@gmail.com>, Kyle Maxwell <krmaxwell@gmail.com>
Cc: Dave Dittrich <dittrich@u.washington.edu>, stix-discussion-list Structured Threat Information Expression/ST <stix-discussion-list@lists.mitre.org>, "Barnum, Sean D." <sbarnum@mitre.org>
Subject: RE: STIX Content Revision and Revocation

A couple things that may not have been clear in the current proposals are:

- All content is native STIX
- When updating a piece of content, the assumption is that you'll resend the entire thing. So if I change the name of a campaign, in the current proposals I would resend you the entire campaign construct with the new name.

Our reasons for this were to allow people to use their existing STIX capabilities to support versioning (no need to understand another format, it's all STIX) and to add as little to STIX as possible. For example, if you want to send just content that was updated instead of an entire document then you would need to either add capabilities to STIX to indicate that certain content was deleted (rather than just omitted because it wasn't updated) or you would need to support some external patch format (likely literal code patches).

Kyle, in your suggestion below are you saying that these patches would be traditional text patches (with + and - in front of lines) or are you saying that STIX should have the capability to support patches natively?

John

From: Terry MacDonald [mailto:terry.macdonald@gmail.com]
Sent: Sunday, February 02, 2014 3:00 AM
To: Kyle Maxwell
Cc: Wunder, John A.; Dave Dittrich; stix-discussion-list Structured Threat Information Expression/ST; Barnum, Sean D.
Subject: Re: STIX Content Revision and Revocation

Hi All,

Isn't the metadata that includes the author and ID of the previous patch the same thing we are talking about? Isn't each revision effectively a 'diff' of the previous with added semantics as to the reason for the change?

Cheers

Terry MacDonald

Terry MacDonald

On 2 February 2014 03:48, Kyle Maxwell <krmaxwell@gmail.com> wrote:
On Thu, Jan 30, 2014 at 7:22 AM, Wunder, John A. <jwunder@mitre.org> wrote:
I think I understand what you're getting at here…my question is how would this be implemented? Since you wouldn't be able to literally use git would that mean all consumers and producers need to support git-like semantics in their tooling? To me that seems a little heavyweight, but maybe I'm missing some factor in how this would be implemented?

That said, even if we don’t use git versioning wholesale it’s possible there are specific concepts in how it works that can be incorporated. You’re certainly right that they’ve spent much more time thinking about versioning than we have.

In my mind, it shouldn't be nearly as onerous to implement diff-like functionality and publish "patches", as it were. So you distribute a patch together with some metadata including the author and the ID of the previous patch (or original source document). Tools can then apply the patch to the existing doc. Alternately, you distribute the new, full doc plus the patch, which on its own specifies everything that changed for an analyst's review.

This should be less difficult to implement than extensive vocabularies and enumerations of possible change types and algorithms to determine how to track these bits.

--
Kyle Maxwell [krmaxwell@gmail.com]
Twitter: @kylemaxwell