Content Repository Interfaces

Adam Halbardier
Booz Allen Hamilton
Supporting NIST

7/16/2012

Automating Content Exchange (Challenges)

- Distribution
  - Local
    - There is no standardized method for accessing and exchanging content
    - Current solutions require manual download and import of content into tools (clients)
  - Distributed
    - No tie exists between content producers and consumers, creating difficulty in managing content revisions
- Reuse
  - Local
    - Identification: providing persistent identifiers for content
    - Searching: identifying appropriate content that already exists
    - Referencing: referencing existing content by id
  - Distributed
    - Searching: enabling searching across multiple, possibly remote, content repositories
    - Referencing: support for identifying or resolving the location of content, in addition to its identifier.

Automating Content Exchange (Challenges) (cont'd)

- Interoperability
  - Local
    - The interfaces for accessing, querying, and managing content are mostly proprietary
    - Many different communications protocols are used
    - XML models organize and support content references using a variety of different methods.
  - Distributed
    - Access to multiple content repositories requires multiple access approaches.
    - Implementing support for a given repository requires support for different:
      - Data models
      - Interfaces
      - Communications protocols
    - Different packaging and compression methods are used by different repositories.

Automating Content Exchange (Challenges) (cont'd)

- Content Revision Management
  - Local
    - Most current content repositories do not support managing revisions of content
    - Most current content repositories do not support retrieval of content by ID and version
    - Local changes to content may alter the intent and breaks integrity.
  - Distributed
    - Caching and replication of content without version awareness causes inconsistencies and conflicts.

Automating Content Exchange – Interface Design (Derived Requirements)

- Standardized interfaces are needed that address content access and exchange (Distribution, Interoperability)
  - Clients must be able to retrieve content from a repository for use (Distribution, Interoperability)
    - Provide a mechanism that enables content in a repository to be referenced by content ID (Reuse)
    - A mechanism must also be provided to specify the version of the identified content (Content Version Management)
    - Content must be retrievable by ID and version. (Content Version Management)

Automating Content Exchange – Interface Design (Derived Requirements) (cont'd)

- Provide a mechanism that enables searching for existing content (Reuse, Interoperability)
- Provide a mechanism for managing content using basic CRUD approaches (Interoperability)
  - On import, the repository must provide a mechanism to detect and resolve content conflicts (different content with the same ID/version)

Basic Assumptions

- Use REST web services (HTTP/TLS)
- Use XML Digital Signatures on content where appropriate
- The repository must handle the complexity of content management so that clients are only responsible for processing the content that is delivered to them.

Interfaces

- Interface 1 (I1) – Retrieve
- Interface 2 (I2) – Search
- Interface 3 (I3) – Metamodel Exchange
- Interface 4 (I4) – Retrieve Top Level IDs (static)

Use case 1: User retrieves content by model ID (I1)

- A user retrieves content based on a known model ID
  - benchmark-key
    - benchmark-id = xccdf_gov.nist_benchmark_USGCB-Windows-7
  - def-key
    - def-id = oval:gov.nist.usgcb:def:123
    - version = 23

I1 – Retrieve

- Purpose
  - To retrieve content from a content repository by ID
- Goals
  - Provide a simple mechanism to retrieve usable content and metadata from a content repository

I1 Query parameters

- Metadata
  - Boolean to indicate the format of the response
  - FALSE – Response should be only the content (default)
  - TRUE – Response should include the content AND metadata
- Depth
  - Integer indicating level of resolution
  - -1 – Unlimited depth (default)
  - 0 – Return only XInclude element
  - 1 – Return top level element populated with XInclude elements
  - etc…

Use case 1: User retrieves content by model ID (I1)

- Request: HTTP GET over TLS
  - https://usgcb.nist.gov/model/benchmark-key/xccdf_gov.nist_benchmark_USGCB-Windows-7?metadata=false&depth=-1
- Response:
  - <xccdf:Benchmark id="xccdf_gov.nist_benchmark_USGCB-Windows-7">
    …
    </xccdf:Benchmark>

Use case 1: User retrieves content by model ID (I1)

- Request: HTTP GET over TLS
  - https://usgcb.nist.gov/model/benchmark-key/xccdf_gov.nist_benchmark_USGCB-Windows-7?metadata=false&depth=0
- Response:
  - <x:include href="https://usgcb.nist.gov/model/benchmark-key/xccdf_gov.nist_benchmark_USGCB-Windows-7"/>

Use case 1: User retrieves content by model ID (I1)

- Request: HTTP GET over TLS
  - https://usgcb.nist.gov/model/benchmark-key/xccdf_gov.nist_benchmark_USGCB-Windows-7?metadata=false&depth=1
- Response:
  - <xccdf:Benchmark id="xccdf_gov.nist_benchmark_USGCB-Windows-7">
    …
    <x:include href="https://usgcb.nist.gov/model/rule-key/xccdf_gov.nist_benchmark_USGCB-Windows-7/xccdf_gov.nist_rule_increase_a_process_working_set"/>
    …
    </xccdf:Benchmark>

Use case 1: User retrieves content by model ID (I1)

- Request: HTTP GET over TLS
  - https://usgcb.nist.gov/model/benchmark-key/xccdf_gov.nist_benchmark_USGCB-Windows-7?metadata=true
- Response:

```
<entity-collection>
        <entity>
                …
        </entity>
        …
</entity-collection
```

Use case 1: User retrieves content by model ID (I1)

- Response (con't)

<entity>
<key>…</key>
<version>…</version>
<property>…</property>
substitution group:
keyed-relationship
composite-relationship
boundary-identifier-relationship
<content>…</content>
</entity>

Use case 1: User retrieves content by model ID (I1)

- Response (con't)

```
<key id="key-id-1">
        <field id="field-1" value="value-1"/>
        <field id="field-2" value="value-2"/>
</key>
```

Use case 1: User retrieves content by model ID (I1)

- Response (con't)

```
<version>1.2.3.4</version>
<property name="prop1">
        <value>val1</value>
        <value>val2</value>
</property>
```

Use case 1: User retrieves content by model ID (I1)

- Response (con't)

<keyed-relationship predicate="http://relationship-uri-1" object="123456"/>

<composite-relationship predicate="http://relationship-uri-2" object="123456"/>

<boundary-identifier-relationship predicate="http://cve.mitre.org" object="CVE-1234-1"/>

Use case 2: Tool retrieves content by global ID (I1)

- What is a global ID?
  - Global ID is an arbitrary ID assigned by the repository to content
  - The global ID in the context of the repository URI is globally unique
- A tool retrieves content based on a known global ID
  - Content repo global ID: 123456789

Use case 2: Tool retrieves content by global ID (I1)

- Request: HTTP GET over TLS
  - https://usgcb.nist.gov/global/123456789?metadata=false&depth=-1
- Response:
  - <xccdf:Benchmark id="xccdf_gov.nist_benchmark_USGCB-Windows-7">
    …
    </xccdf:Benchmark>

Discussion

Use case 3: User wants to find existing content (I2)

- User needs a robust mechanism to find existing content across repositories

I2 – Search

- Purpose
  - Query into repository to find existing content
- Goals
  - Provide a robust query language that allows querying of metadata
  - Return metadata of discovered content

Use case 3: User wants to find existing content (I2)

- Possible near-term solution:
  - Build a robust query language that can be marshalled/unmarshalled to/from XML and/or JSON
  - Support querying on arbitrary attributes and relationships

Use case 3: User wants to find existing content (I2)

```
selectEntitiesWith(
    allOf(
        anyOf(
            key(
                "http://scap.nist.gov/resource/content/source/1.2#key-datastream-collection",
                field("datastream-collection-id", "scap_gov.nist_collection_Win7-54-1.2.0.0.zip")
            ),
            contentId("123456789")
        ),
        entityType("http://scap.nist.gov/resource/content/source/1.2#document-datastream-collection"),
        relationship(
            anyOf(
                relationshipType("http://scap.nist.gov/resource/content/source/1.2#relationship-datastream-boundary"),
                relationshipType("http://scap.nist.gov/resource/content/source/1.2#relationship-component-boundary"),
                to(
                    contentId("23456")
                )
            )
        )
    )
);
```

Use case 3: User wants to find existing content (I2)

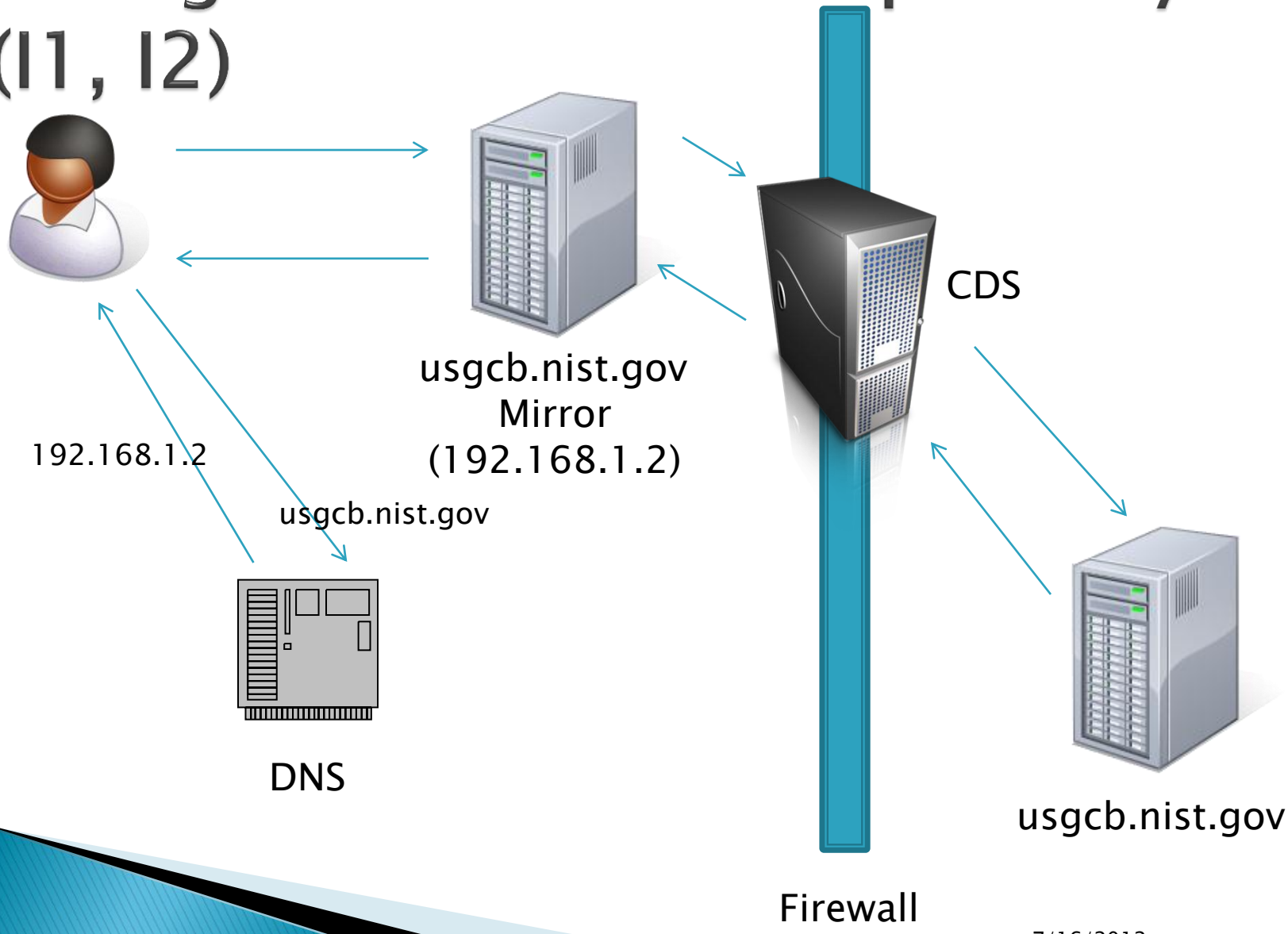
- Possible long-term solution:
  - Build a search engine for content repositories

Discussion

Use case 4: Administrator needs to configure a mirrored repository (I1, I2)

- In cases where a client cannot reach a content repo (e.g. an air-gap network), an administrator may set up a repository mirror
- Example:
  - Client attempts to reach repository at http://usgcb.nist.gov, but it cannot access that domain
  - Administrator configures client (or DNS server) so that when client attempts to resolve http://usgcb.nist.gov it resolves to a discoverable repository
  - The discoverable repository proxies requests to http://usgcb.nist.gov, or serves up a cached copy of the content

Use case 4: Administrator needs to configure a mirrored repository (I1, I2)



Discussion

Use case 5: User needs the context for retrieved metadata (I3)

- User retrieves content with metadata using I1
- User needs the context of the metadata

I3 – Metamodel Exchange

- Purpose
  - Exchange a content repo metamodel
- Goals
  - Provide a simple interface to enable requesting of a content repo metamodel by revision

Use case 5: User needs the context for retrieved metadata (I3)

- Request: HTTP GET over TLS
  - https://usgcb.nist.gov/metadata/metamodel-1/12
- Response:
  - The metamodel

Discussion

Use case 6: Repository needs to cache content (I4)

- In the CDS scenario, if the CDS did not exist then the mirror would need to cache ALL the usgcb.nist.gov content
- Solution:
  - Repositories provide a static URL to retrieve all top level entity global IDs

I4 – Retrieve Top Level IDs

- Purpose
  - Permit crawling of the content repo data
- Goals
  - Provide the starting points from which a client could retrieve all of the data in a content repo

Use case 6: Repository needs to cache content (I4)

- Request: HTTP GET over TLS
  - https://usgcb.nist.gov/all
- Response:

```
<top-entities>
        <gid>123456789</gid>
        <gid>123456788</gid>
        <gid>123456777</gid>
        …
</top-entities>
```

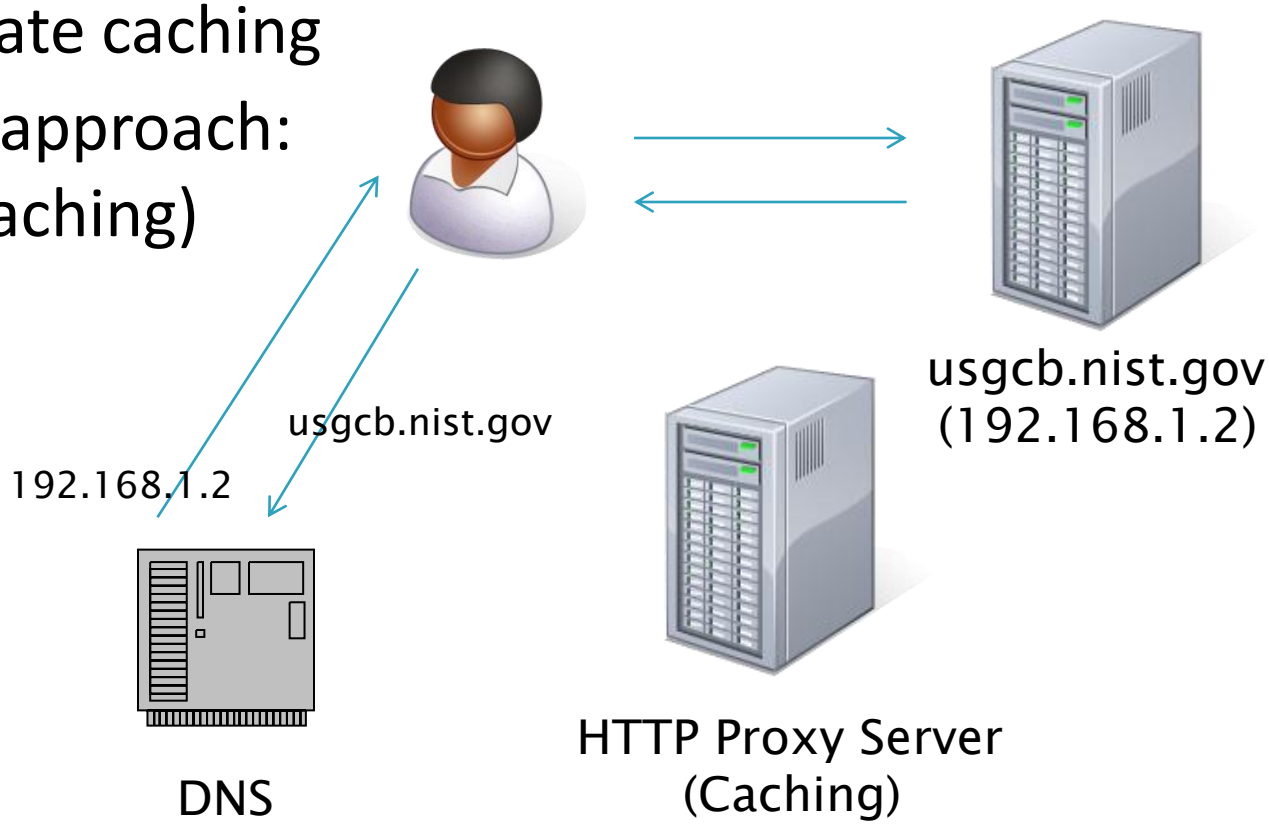
Discussion

Issue: Caching and Security (Why?)

- Basic assumption: caching and secure communications is necessary
- Caching:
  - Reduce network load
  - Increase availability
  - Increase performance
- Security
  - Integrity of content
  - Trustworthiness of content
  - Confidentiality of content

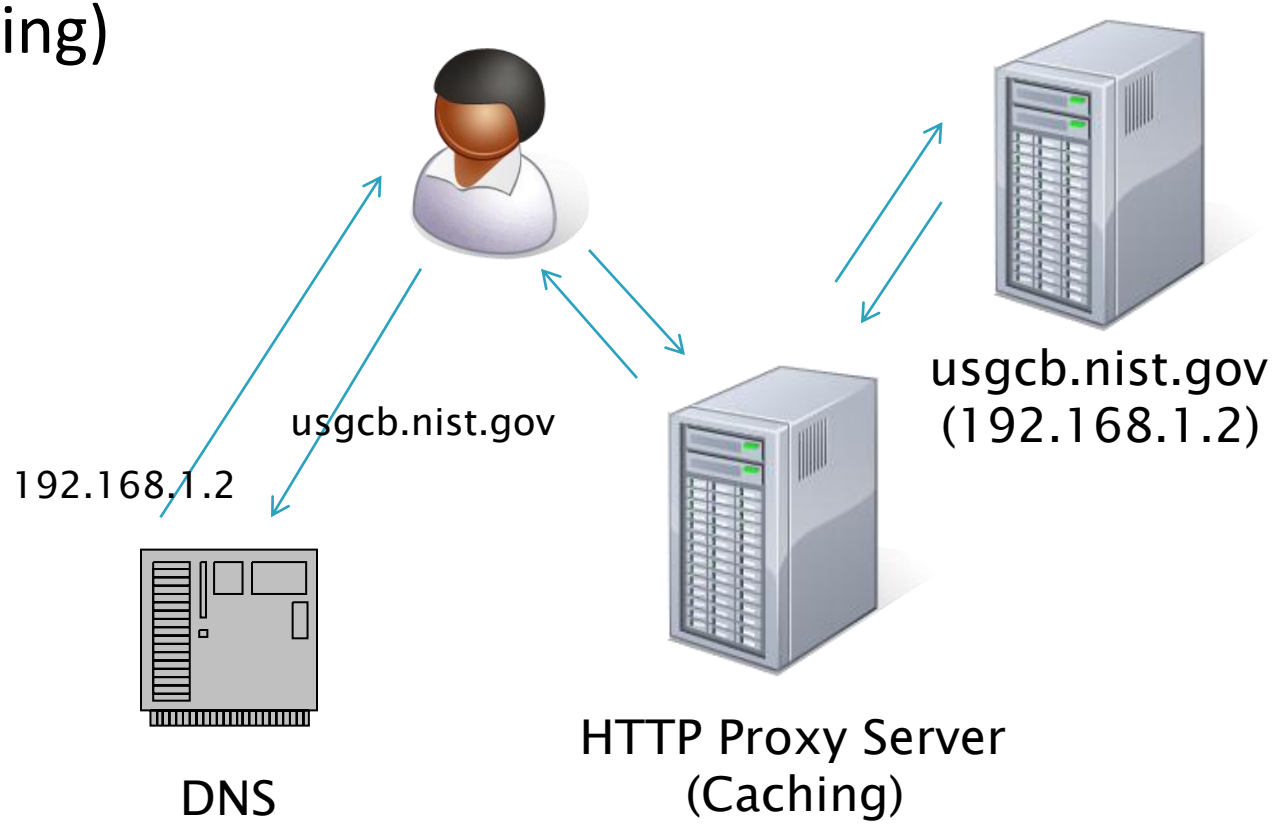
Issue: Caching and Security (Option 1)

- The proposed approach to interface 1 does not easily facilitate caching
- Basic approach: (No caching)



Issue: Caching and Security (Option 1)

- Basic approach: (Caching)



Issue: Caching and Security (Option 1)

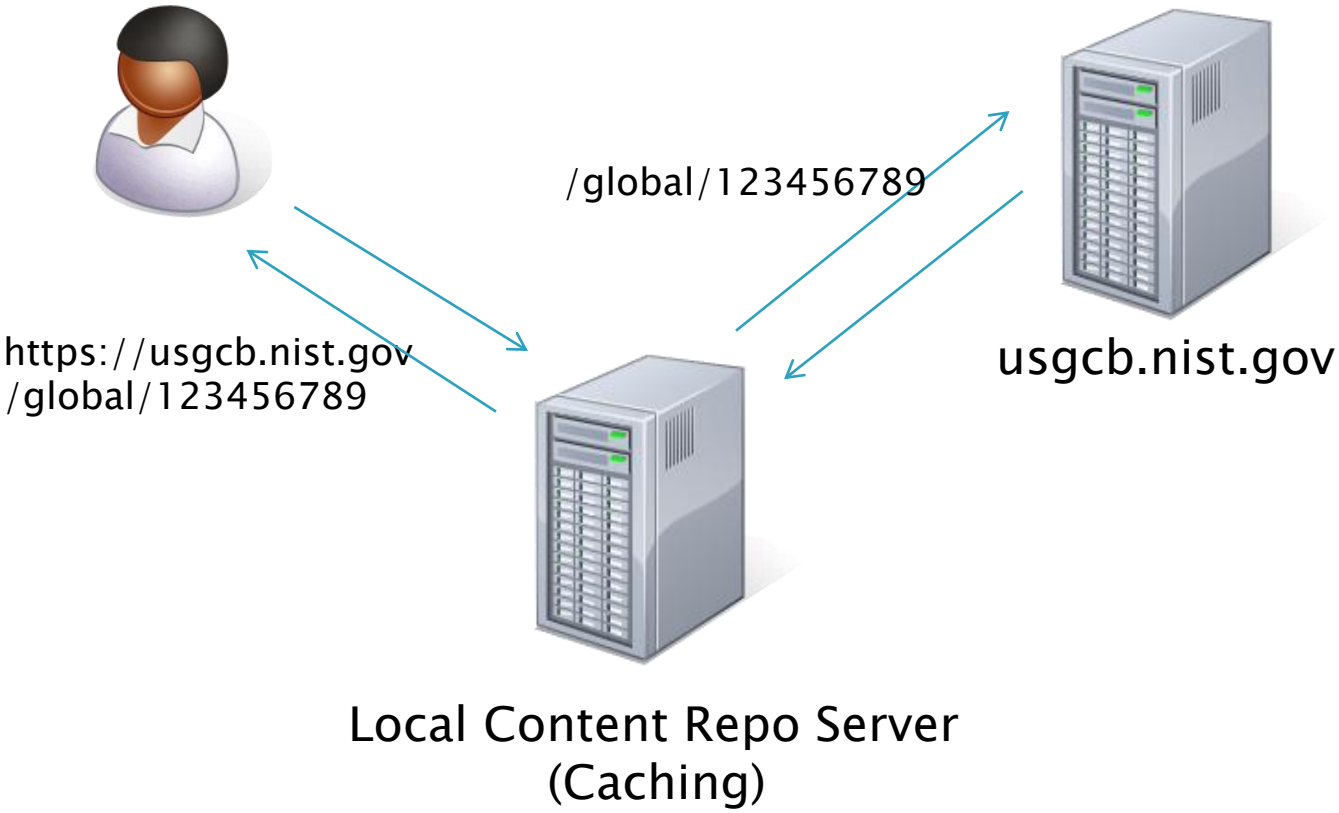
- Advantages:
  - The caching pattern is simple
  - Tools already exist to support the capability
  - Relies on existing technology

Issue: Caching and Security (Option 1)

- Issue with the caching approach:
  - Does not support TLS if the secure connection is established between the client and the repo, through the HTTP Proxy
    - Why? The packets cannot be replayed
  - Does support TLS if 2 secure connections are established; 1) between the client and the proxy server, and 2) between the proxy server and repo
    - The client knows that the proxy is decrypting the data
    - The proxy will likely use a local cert that the client will need to trust
    - Likely difficult to acquire approval to configure
  - All resources must be requested individually

Issue: Caching and Security (Option 2)

- Local content repo server approach:



Issue: Caching and Security (Option 2)

- Advantages:
  - The content repo server is content aware, so batch requests can be supported
  - Secure TLS communication are natively supported as the client is always establishing a connection with the local repository
  - More complex caching strategies may be employed

Issue: Caching and Security (Option 2)

- Disadvantages:
  - Lose the simplicity of using HTTP proxy caching (the capability already exists)
  - Requires additional functionality be built into the content repositories
  - Requests are more complex as a simple HTTP GET is not sufficient

Issue: Caching and Security (Discussion)

- Is caching and secure communications necessary?
- Which approach is preferred? Option 1, 2, or other?

Issue: Maintaining the Metamodel

- The metamodel defines how to extract metadata from content
- Issue:
  - How do we deal with metadata extracted using different metamodels?
  - How do we exchange metamodels?
  - How do we link metadata to the metamodel that was used to produce it?

Issue: Maintaining the Metamodel

- Proposal:
  - Each content repository may have 1 or more metamodels
  - A metamodel in a repo may only be updated in a “backward compatible” manner (i.e. rules may be added, but existing rules may not be changed or removed)
  - Metadata communicated over I1 must also include the name of the metamodel used to produce the metadata
  - I3 is leveraged to retrieve a metamodel
  - When content is processed in a repository, it is parsed through the latest of all of the active metamodels in the repository

Issue: Maintaining the Metamodel

- Proposal (con't):
  - When a request for metadata occurs over I1, the metadata set requested is specified in the URI
    - https://usgcb.nist.gov/model/benchmark-key-metamodel-1/xccdf_gov.nist_benchmark_USGCB-Windows-7?metadata=false&depth=-1
  - Assumption: IDs are unique in all models

Issue: Maintaining the Metamodel (Discussion)

- Is the solution reasonable?
- Additional thoughts, questions, concerns?

Conclusion

- Way forward:
  - Incorporate feedback into proposals
  - Implement functionality into content repository reference implementation
  - Write content repository interface specifications