Merkle Tree Authentication of HTTP Responses

Roberto J. Bayardo
IBM Almaden Research Center
San Jose, CA
bayardo@alum.mit.edu

Jeffrey Sorensen
IBM Watson Research Center
Yorktown Heights, NY
sorenj@us.ibm.com

ABSTRACT

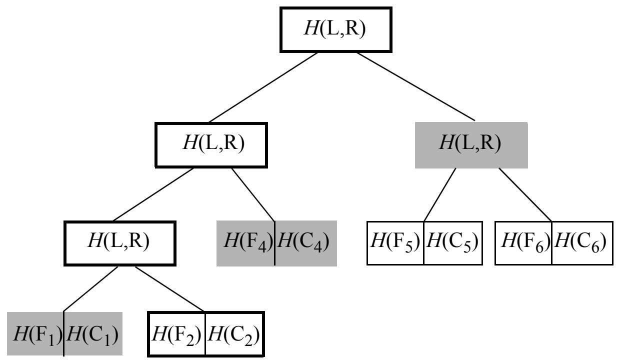
We propose extensions to existing web protocols that allow proofs of authenticity of HTTP server responses, whether or not the HTTP server is under the control of the publisher. These extensions protect users from content that may be substituted by malicious servers, and therefore have immediate applications in improving the security of web caching, mirroring, and relaying systems that rely on untrusted machines [2,4]. Our proposal relies on Merkle trees to support 200 and 404 response authentication while requiring only a single cryptographic hash of trusted data per repository. While existing web protocols such as HTTPS can provide authenticity guarantees (in addition to confidentiality), HTTPS consumes significantly more computational resources, and requires that the hosting server act without malice in generating responses and in protecting the publisher's private key.

Categories: H.3.5 [**Online Information Services**]: *Data Sharing*
General terms: Security
Keywords: Merkle hash tree, web content distribution, authenticity

1. MERKLE TREES FOR WEB CONTENT

Merkle hash trees [5] have been proposed as a model for authenticating query results from untrusted database servers using only a small amount of trusted information [3]. One can view each web repository as a database, and each HTTP GET request a query for an individual resource. We explore the feasibility of authenticating both 200 (OK) and 404 (NOT FOUND) responses from untrusted web servers through appropriately exploiting Merkle hash trees over web repositories. The appeal of Merkle trees is that they require only that the root hash value be obtained over a trusted channel in order to authenticate arbitrary requests to a particular repository. We believe this property makes them applicable for efficient authentication of most HTTP GET responses, excepting only those involving highly dynamic content.

For a given site, our scheme involves computing a Merkle tree structure that consists of hashes of both resources and their *canonical web paths*. A canonical web path is a normalized form of the resource URL whereby the scheme and hostname portion are removed, escape sequences decoded, and so on. Its arrangement is as follows: leaf nodes represent resources, and store cryptographically secure hashes of the resource itself and its canonical web path. The value of a leaf node is the result of hashing these two hash values using a cryptographically secure binary hash function. Leaf nodes are ordered from left to right according to the canonical web path hash value (order can be imposed by treating this value as an unsigned integer.) This ordering provides a mechanism for generating a proof that a specified path name is not present in the directory without disclosing the path names of the files that are present. Internal nodes consist of pointers to left and right children that enforce the ordering. The value of each internal node is the result of hashing the values of its left and right children. The figure in the next column depicts such a tree over a site consisting of five resources F_1, F_2, F_4, F_5 and F_6.



The client must obtain the tree's root hash value from a trusted source, and thereafter it need only verify that each response is consistent with this root value in order to determine response authenticity. For example, consider the case where the client requests the resource identified by canonical web path C_2 in the figure. The *authentication path* of this resource is the set of nodes from the resource leaf to the root, and is depicted by the nodes with bold outlines. In addition to returning the resource F_2, in order to support 200 (FOUND) response authentication, the server must return the values of the indicated children of each node on this path, as depicted by shading. This *auxiliary authentication information* and the resource itself provides the client with all information necessary to compute the values along the authentication path from leaf to root. If the computed root hash matches that obtained from the trusted source, the response is authentic. Attempts by the server to corrupt either the response body or the auxiliary authentication information will be detected, assuming hashing is secure.

Next, consider the case where the client issues a request for some resource identified by canonical web path C_3 that does not exist in the site. To support authentication of the resulting 404 (NOT FOUND) response, the server must somehow prove that there is no such resource present. This is accomplished by returning values along the two authentication paths that originate at the leaves whose canonical web path hashes immediately precede and follow the value $H(C_3)$ in the before-mentioned leaf ordering. In our example, we assume these two leaves are those for resources F_2 and F_4. The server must thus return $H(F_2)$, $H(C_2)$, $H(F_4)$, $H(C_4)$, and the values of the children of nodes along the two authentication paths. To prove authenticity, the client verifies that both of the authentication paths are consistent with the root hash as in the 200 (OK) case above. Additionally, it must verify that (1) $H(C_2) < H(C_3) < H(C_4)$ and (2) no other leaf exists between the leaves that terminate the authentication paths based on their right-left structure.

Auxilliary authentication information can be returned Base64 encoded through a special HTTP response header. For both the 200 and 404 cases, the amount of such information is proportional to the height of the Merkle tree, which is logarithmic in the number of resources assuming the tree is balanced.

2. ROOT HASH DISTRIBUTION

A remaining problem is that of securely delivering the root hash value, which authenticates all of the content. Already established

WWW 2005, May 10-14, 2005, Chiba, Japan.
1-59593-051-5/05/0005

web protocols provide three elegant methods for distribution of this small item. These techniques are not mutually incompatible, and may all be simultaneously used in accordance with a particular client's policies.

DNS-SEC: The DNS system, with its distributed automatic caching, dynamic update capability, and design for small data records would seem to be an excellent mechanism for publishers to use to distribute the root hash value. The hash could be published using either the existing TXT DNS records, or through a format extension to a new type of DNS record. Unfortunately, DNS itself lacks authentication mechanisms. While this has not made the internet unusable, it certainly is one of its most cited weakness. Fortunately, DNS-SEC, the next generation of DNS, incorporates public-key based digital signatures to authenticate the data records returned by each query. No modification of existing protocols is needed to support distribution of the root hash as a specially formatted TXT DNS record, and the DNS time-to-live (TTL) property naturally supports expiration of root hashes to handle repository updates.

Unfortunately, DNS-SEC remains relatively undeployed at the time of this writing. We therefore present two other approaches which may be more feasible in the near term as DNS-SEC matures.

HTTPS to Content Provider Server: Transport layer security (TLS) is an approach for authenticating and securing TCP/IP communications through public-key cryptography. TLS is the basis of the HTTPS web protocol, thus HTTPS itself could be used as a distribution mechanism for root hash information. This approach must involve a standardized scheme for transforming the URI for the requested resource into the name of a trusted server for obtaining the root hash. This is because if a URI of http://www.popularsite.org/ is mapped to multiple mirror servers by multiple DNS A records, then requests for https://www.popularsite.org/ would also connect to these servers, which we presume lack sufficient trust to answer HTTPS requests on behalf of the publisher. A standardized hostname extension such as httpa.www.popularsite.org could be created with a DNS A records that point only to the publisher's servers, and which have the appropriate TLS credentials to allow HTTPS connections.

Certified PKI Signed Root Hash: Distributing the root hash as a file is subject to the attack of a malicious mirror altering content arbitrarily, computing the new root hash, and serving the malicious root hash instead of the true root hash. This attack can be prevented if the root hash is digitally signed using the publisher’s private key, the public key of the publisher is made available, and the public key of the publisher is certified by one of the certificate authorities recognized by the client browser. While this technique can authenticate the content, it cannot assure the freshness of the content. This can be ameliorated through the use of digitally signed validity periods. The protocol could specify a standard location within a mirrored repository where the signed root hash information is to reside.

3. IMPLEMENTATION OVERVIEW

We now sketch the roles and responsibilities needed to implement the proposed protocol in practice and end with a summary of protocol overhead.

Content originators: The primary responsibilities of a content originator include computing and maintaining the Merkle hash tree for its web resources and distributing its root hash value through one or more of the previously outlined methods of distribution. We note that each site update results in a new root hash value. While this property prohibits using the protocol for highly dynamic content, even dynamic content typically references relatively static images that could be effectively mirrored and authenticated through the proposed protocol.

Mirror hosts: Mirror sites must download site content from the originators or other up-to-date mirrors, and recompute or download the changes to the hash tree structure. Hash trees can be maintained incrementally so that each resource modification requires only distributing or recomputing hash information along the authentication paths of any added or removed leaves.

In response to HTTP requests, for any user agent capable of executing the authentication protocol, mirror hosts should return the auxiliary authentication information through a special HTTP response header in addition to the standard HTTP response fields and body.

Web Browser Clients: Browsers must obtain the necessary root hash values and verify authenticity through the root hash and the auxiliary authentication information as has been previously described. Browsers should also provide configurable policies in dealing with authentication failures. In cases where many mirror sites are available, it is plausible for the browser to be configured to automatically fail over to an alternate site. Since authentication failures can be the result of out of date root hash values or mirror content rather than malicious tampering, the policy should also dictate how root hash version conflicts should be resolved. Policies should finally specify the handling of non-200 or 404 responses in a manner compatible with the security requirements of the client. Redirect requests, for example, could be returned by a malicious server to prevent access to desired content. Clients implementing the protocol might foil such an attack by specifying that redirects are not a valid response to any request intended for authentication.

Overhead summary: The protocol’s impact on the performance of web servers is small: beyond the overhead required of HTTP, it involves only the delivery of a limited amount of precomputed information per request. Web clients have a few additional computational burdens, including computing cryptographically secure hashes over response bodies in order to recreate the root hash, and obtaining the root hash of each repository from a trusted source. Because client-side compute resources are typically abundant, the overhead of hash computation is unnoticeable, especially compared to the overhead of rendering. The overhead of retrieving root hashes over secure channels is also small since only one request per repository is required during the time-to-live period of the root hash value. Even time-to-live values of a few minutes are feasible without significant effects on browsing performance.

4. CONCLUSIONS

We believe there is an immediate need for authentication schemes that support untrusted web serving infrastructures. Our experiences suggest that a Merkle tree-based protocol is both a feasible and practical solution to this problem. While we have only touched upon some of the issues necessary for deploying Merkle hash tree based authentication schemes for web content in this short paper, an extended technical report is available [1] that provides more details, including: pseudo-code for the verification procedures, a deeper look at Merkle tree maintenance, and experiments and experiences with an actual implementation.

5. REFERENCES

[1] R. J. Bayardo and J. Sorensen. Mirrors and Authenticity: an Enhanced Protocol for Content Delivery. IBM Research Report RJ 10335, 2004.

[2] R. J. Bayardo, A. Somani, D. Gruhl, and R. Agrawal. YouServ: A Web Hosting and Content Sharing Tool for the Masses, In *Proc. of WWW-2002*, 2002.

[3] A. Buldas, M. Roos, J. Willemson. Undeniable replies for database queries. In *Fifth Int’l Baltic Conference on DB and IS*, 2002.

[4] M. J. Freedman, E. Freudenthal, D. Mazieres. Democratizing Content Publication with Coral. In *Proc. 1st USENIX/ACM Symposium on Networked System Design and Implementation*, 2004.

[5] R. C. Merkle. Protocols for public key cryptosystems. In *Symposium on Security and Privacy*, 122-134, 1980.