

Publius

A Robust, Tamper Evident, Censorship Resistant WWW Based Publishing System



By Lorrie Cranor AT&T Labs
Avi Rubin Marc
Waldman New York University

Proc. 9th USENIX Security Symposium, 2000
Presented by Anyi Liu
Dec. 2, 2004

Acknowledgments



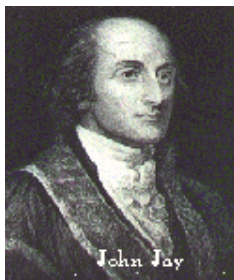
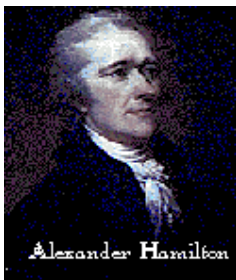
- Some of the following slides are adapted from the slides created by the authors of the paper, and the following link
- <http://www-net.cs.umass.edu/cs791n/marc.ppt>

Agenda

- Motivations and Design Goals
- Main Ideas
- Implementation Details
- Security Challenges

What is Publius in History?

- Publius is the *pen name* used by three authors of Federalist Papers
- Authors may publish their papers without worrying about been disclosed of real-world identities.





Design Goals

- Censorship resistant
 - Difficult for a third party to modify or delete content
- Tamper evident
 - Unauthorized changes should be detected
- Source anonymous
 - No way to tell who published the content
- Updateable
 - Changes or deletion of content should be possible for publishers



Design Goals (cont)

- Deniable
 - Involved third parties should be able to deny knowledge of what is published
- Fault Tolerant
 - System remains functional, even if some third parties are faulty or malicious
- Persistent
 - No expiration date on published materials



Related Works

- Connection Based Anonymity
 - Hide identity of requestor
 - Anonymizing proxies (for example Anonymizer.com)
 - Freedom (Zero-Knowledge Systems)
 - Crowds (AT&T Labs-Research)
- Location or Author Based Anonymity
 - Hide identity of author or WWW server
 - USENET Eternity System
 - Freenet
 - Intermemory
 - Rewebber



Agenda

- Motivations and Design Goals
- Main Ideas
- Implementation Details
- Security Challenges



Theoretical Foundation

- Shamir secret sharing theorem
 - Suppose Alice wants to share a secret among n agents. Any subset of n agents (say, k) can use their shares to reconstruct the secret
 - No subset of size $< k$ learns anything
 - Assume that up to $n-k$ agents may be “bad”, and may not reveal their shares. The rest of the agents are “good”, and follow the protocol
 - The bad agents can't prevent the good agents from reconstructing the secret



Publius System Roles

Publius is a Client-Server paradigm

- Publishers
 - Post Publius content to the web
- Servers
 - A set of hosts that store random-looking content
- Retrievers
 - Browse Publius content on web



Publius System Operations

There are basically four type of operations

- Publish
 - A publisher posts content across multiple servers in a source-anonymous fashion
- Retrieve
 - A retriever gets content from multiple servers
- Delete
 - The original publisher of a document removes it from the Publius servers
- Update
 - The original publisher modifies a document



Publius Publish Operation

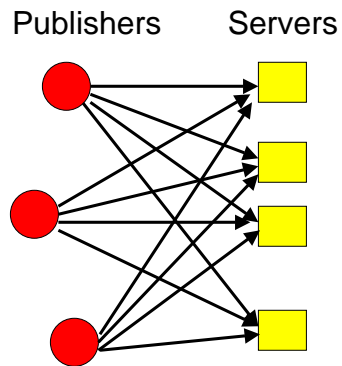
- Alice generates a random *symmetric* key K
- She encrypts message M with key K , producing $\{M\}_K$
- She splits K into n shares, using *Shamir secret sharing theorem*, such that any k can reproduce K
- Each share is uniquely named:

$$name_i = wrap(H(M \cdot share_i))$$

↑
MD5

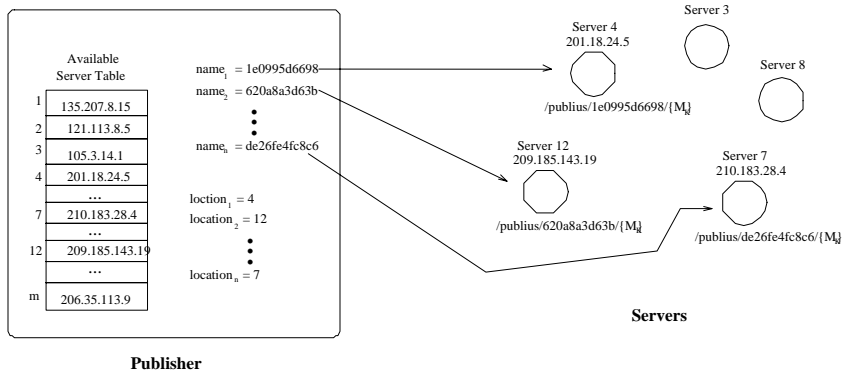
Publius Publish Operation(Cont')

- A set of locations is chosen:
 $location_i = (name_i \text{ MOD } m) + 1$
- Each $location_i$ indexes into the list of m servers
- If $\text{sizeof}(\text{location}) < d$, start over again
- Otherwise, Alice publishes $\{M\}_K$ and $share_i$ into a directory $name_i$ on the server at $location_i$
- A URL containing at least the d $name_i$ values is produced



Note: d represents the minimum number of unique server that will hold the Publius content.

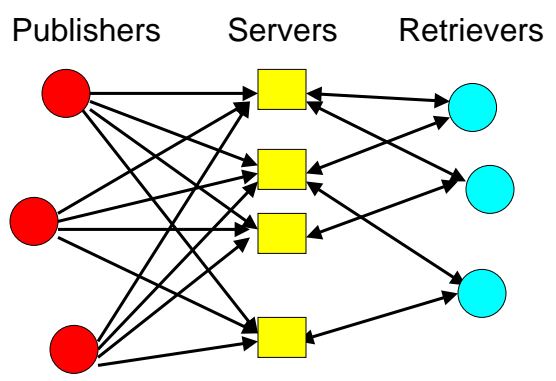
Publius Publishing



Publius Retrieve Operation

- Bob parses out each $name_i$ from URL, and for each, computes:
$$location_i = (name_i \text{ MOD } m) + 1$$
- Bob chooses k of these, and retrieves the encrypted file $\{M\}_K$ and $share_i$ at each server
- Bob combines the shares to get K , and decrypts the message M
- Bob verifies that each name value is correct:
$$name_i = wrap(H(M \cdot share_i))$$
- If $name_i$ can't be reconstruct through M , $share_i \rightarrow$ The content has been tampered.

Retrieving a Publius document





Publius Delete Operation

- Alice generates a password PW when publishing a file
- Alice includes $H(\text{server_domain_name} \cdot PW)$ in server directory when publishing
 - Note that each server does not store $PW/H(PW)$, because it prevents malicious server operator from deleting content on all other servers
- Alice deletes by sending $H(\text{server_domain_name} \cdot PW)$ and $name_i$ to each of the n servers hosting content



Publius Update Operation

- Idea: change content without changing original URL, as links to that URL may exist
- In addition to the file, the share, and the password, there may be an update file in the $name_i$ directory
- This update file will not exist if Alice has not updated the content



Publius Update Operation(Cont')

- To update, Alice specifies a new file, the original URL, the original password PW , and a new password
- First, the new content is published, and a new URL is generated
- Then, each of the n old files is deleted, and an update file, containing the new URL, is placed in each $name_i$ directory of each servers



Publius Update Operation (Cont')

- When Bob retrieves updated content, the server returns Bob the updated URL
- Bob receive all the updated URL from k servers, and checks if all of the new URLs are identical
- If yes, Bob will retrieves the content at the new URL



Other Features

- Entire directories can be published by exploiting the updateability of Publius
- Mechanism exists to encode MIME type into Publius content
- Publius URLs include option fields and other flags, the value of k , and other relevant values
 - Older browsers prohibit URLs of length >255 characters
 - Once this limitation is removed, URLs can include server list, making this list non-static

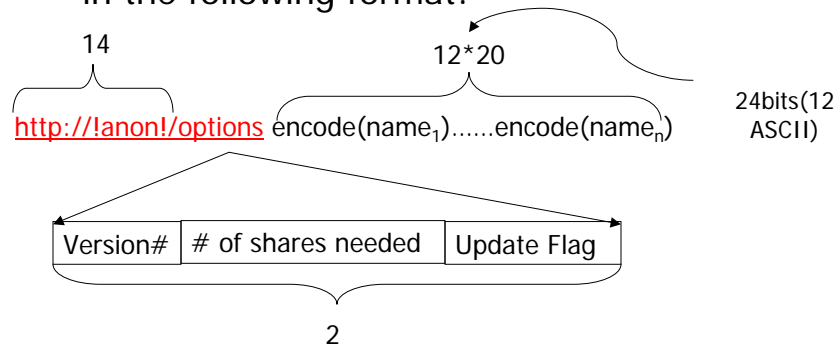


Agenda

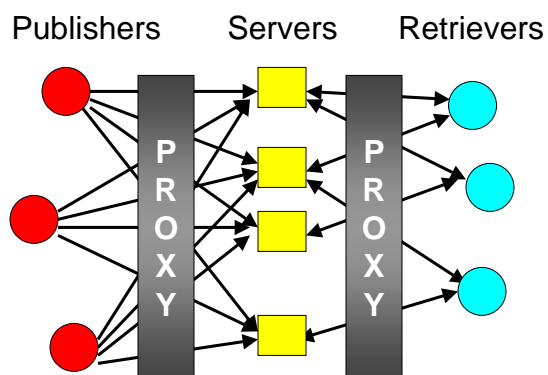
- Motivations and Design Goals
- Main Ideas
- Implementation Details
- Security Challenges

Publius URLs

- Since most old browsers only accept URL with at most 256 characters, Publius defines URL in the following format:



Publius proxies



- Publius proxies running on a user's local machine or on the network handle all the publish and retrieve operations
- Proxies also allow publishers to delete and update content



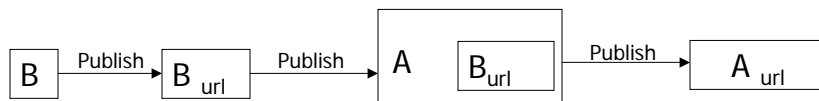
Server/Client Side Software

- Server: Accept HTTP POST operation
 - Requested operation, file name, password, and other info are passed through POST request
- Client=Http Proxy + a set of publish tools
 - Return values:
 - Success
 - Unable to find M
 - Update → Re-direct



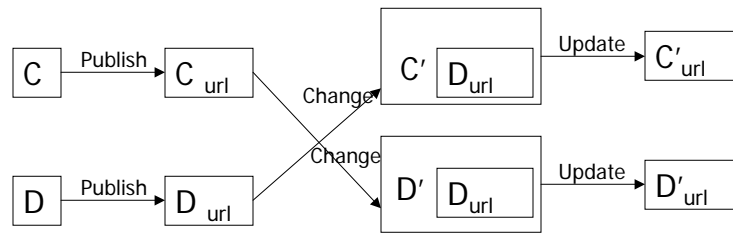
Publish Mutual Hyper-Links

- How to publish documents contains hyper-links. Let's discuss two cases:
- Case1: Alice trying to publish HTML file A and B, A contains a hyper-link to file B



Publish Mutual Hyper-Links(cont')

- Case2: Alice trying to publish HTML file C and D, C contains a hyper-link to file D, and D contains C



Agenda

- Motivations and Design Goals
- Main Ideas
- Implementation Details
- Security Challenges



Share deletion or corruption

- Share deletion or corruption
 - If all n copies of a file, or $n-k+1$ copies of the shares, are deleted, then the file is unreadable (less than k shares exist)

Solution: Increasing n , or decreasing k , makes this attack harder



Update file deletion or corruption

- Update file deletion or corruption (case1)
 - If there is no update file, malicious server operator, Mallory, could create one and pointing to bad content
 - This requires the accomplice of at least k other server.

Solution:

- This attack motivates a higher value of k
- If *update* flag was turned off, it will prevent this attack



Update file deletion or corruption(cont')

- Update file deletion or corruption (case2)
 - If Publius content has already been updated, Mallory must corrupt update files on $n-k+1$ servers
 - Of course, if Mallory can do this, she can censor any document
 - Larger n and smaller k make this more difficult



Update file deletion or corruption(cont')

- Update file deletion or corruption (case3)
 - If Mallory can delete the updated files on all servers to be deleted, he can restore the content to its previous state
 - This motivate client to retrieve from all n copies before perform verification
 - This solution seems impractical
- Deciding upon good values for n and k is difficult
 - Case1, 2 vs Case3
 - No suggestions from Waldman et al.



Denial of service attacks

- Publius, like all internet services, is subject to DoS attacks
 - Flooding is less effective, as $n-k+1$ servers must be attacked
 - A malicious user could attempt to fill disk space on servers
 - Some mechanisms in place to prevent this



Threats to publisher anonymity

- If the Publius content contains any identifying information, anonymity will be lost
- Publius does not provide any connection based anonymity. Eavesdrop is possible.
 - If you act as a publisher, you must anonymize your connections with the Publius servers



“Rubber-hose cryptanalysis”

- The technique of breaking a code or cipher by finding someone who has the key and applying a rubber hose vigorously and repeatedly to the soles of that luckless person's feet until the key is discovered.
- Even though some server could be forced to delete Publius content, to do it across the countries and jurisdictions is very expensive and impractical



Publius vs Freenet

- Both provide publisher anonymity, deniability, and censorship resistance
- Freenet provides anonymity for retrievers and servers, as well
 - Cost is high: data must be cached at many nodes
- Publius provides persistence of data
 - Freenet does not
 - Can any p2p system provide persistence?



Questions

- How do you publish Publius URLs anonymously?
 - The first person to publish a Publius URL must have some connection with the publisher of the content
 - If you have somewhere secure and anonymous to publish the Publius URLs, why do you need Publius?
 - One possible answer: censorship resistance
 - But server operators are then potentially liable



Questions

- How deniable is Publius?
 - Publius URLs are public
 - With minimal effort, a Publius server operator could determine the content being served



Questions

- Could Publius be made into a p2p service?
- Would it be appropriate to do so?



More questions?
