USENIX Association

Proceedings of the 17th Large Installation Systems Administration Conference

San Diego, CA, USA
October 26–31, 2003

DryDock: A Document Firewall

Deepak Giridharagopal – The University of Texas at Austin

ABSTRACT

Auditing a web site's content is an arduous task. For any given page on a web server, system administrators are often ill-equipped to determine who created the document, why it's being served, how long it's been publicly viewable, and how it's changed over time.

To police our web site, we created a secure web publishing application, DryDock, that governs the replication of content from an internal, developmental web server to a stripped-down, external, production web server. DryDock codifies a formal approval process that forces management to approve all web site changes before they are pushed out to the external machine. Users never interact directly with the production machine; DryDock updates the production server on their behalf. This allows administrators to operate their production web server in a more secure and regimented network environment than normally feasible.

DryDock audits documents, tracks revisions, and notifies users of changes via email. Managers can approve files for publication at their leisure without the risk of inappropriate content ever being publicly visible. Web authors can develop pages without intimate knowledge of security policies. And administrators can instantly know the complete history of any file that has ever been published.

Introduction

Information is valuable. While nearly all organizations go to great expense to protect their networks, a much smaller percentage have formal safeguards against the accidental dissemination of sensitive information. In a web server environment where many users can update different parts of a web site at once, auditing the server for inappropriate content becomes an increasingly difficult system administration task. Most system administrators can't easily determine why a particular file exists on a web server, or who in management authorized its publication. How can administrators be expected to safeguard information if they can't tell which documents are fit for the public and which ones aren't?

This was the situation in our organization, Applied Research Laboratories, The University of Texas at Austin (ARL:UT).

Motivation

Since 1994, ARL:UT has had a publicly accessible web server. Initially, we served simple, static pages. There was relatively little web server traffic, and, like many organizations at the time, we didn't concentrate on the security of our network or our information. Our web server resided on our internal network with full access to our file server and other intranet resources, and employees were trusted to only serve documents that were appropriate for public viewing.

By 2001, our web presence had grown in both traffic and size by several orders of magnitude. Our site's much larger scale made it impossible for administrators to effectively police its content. Our formal publishing policy and guidelines were conceived for paper documents, not web pages. Their checks and balances were inadequate when applied to our web architecture, which allowed users to publish pages without even a cursory review. Many staff members were unaware that web pages even fell under these guidelines. We found ourselves unable to track who published specific files and who had deemed those files fit for the public. Since much of ARL:UT's research is sensitive and proprietary, we needed to strictly regulate the flow of information from inside our organization onto our public web server.

To ensure that only material suitable for public viewing appeared on our web site, we needed to force documents to undergo an approval process – only files that successfully complete the process will move to the web server. Furthermore, for any publicly viewable file, we needed a way to determine who authorized its publication, when the file was published, and for what reason. Not only did we need this information available for currently published files, but for any previous versions as well. A web publishing system that provided us with these features would enforce our information security policies by ensuring publicly viewable content is acceptable and well accounted for.

Due Diligence

In late 2001, we searched for tools that would put our new web publishing plan into service. Of the countless managed web publishing solutions on the market at the time, we found none that, out-of-the-box:

- implemented a role-based approval process appropriate for our organization's managerial structure
- gave us the thorough revisioning and auditing capabilities we needed

- didn't mix approved and unapproved content on our public web server
- had a friendly, web-based user interface that managers could understand
- were minimally intrusive for both our system administrators and our web developers
- used technologies we were already familiar with

Sweeping, monolithic content management systems such as Vignette StoryServer [17] were inappropriate for our environment. ARL:UT is comprised of many autonomous groups that have their own web development methods and practices. While the need for a formal information security process was universally recognized, a sweeping content management system was both politically infeasible and far too expensive.

Portal and weblog systems such as PostNuke [7], Tiki [20], or Plone [8] focus on creating highly dynamic, interactive web sites. Thus, they frequently offer collaborative features such as article syndication, Wiki[1] systems, forums, and user commentary. At ARL:UT, however, we needed a strict web presence that was decidedly static and non-collaborative – this obviated many of these systems' features. All we wanted was software that would allow approved documents through to the web server, while blocking all other unauthorized updates. All of these packages required so much customization that it was easier for us to build our own solution, tailored specifically to our environment.

[1] "Wiki Wiki Web is a set of pages of information that are open and free for anyone to edit as they wish, through a web interface. The system creates cross-reference hyperlinks between pages automatically. Anyone can change, delete, or add to anything they see." [10]

Having resigned ourselves to writing a custom tool, we began looking at platforms upon which we could base our application. We were particularly interested in the Zope [16] application server. Written in Python, Zope has been used to build many complex and dynamic web sites. Its features include user management, web-based administration, searching, clustering, and syndication. Like the aforementioned packages, however, a great deal of Zope's dynamic componentry was of no use to us, and much of Zope's functionality fell far outside the scope of simple publication oversight. Though these issues weren't intractable, when combined with Zope's steep learning curve, they led us to look at other less complex and less ambitious platforms.

We settled on WebKit, the Webware for Python [22] application server. WebKit uses a design pattern

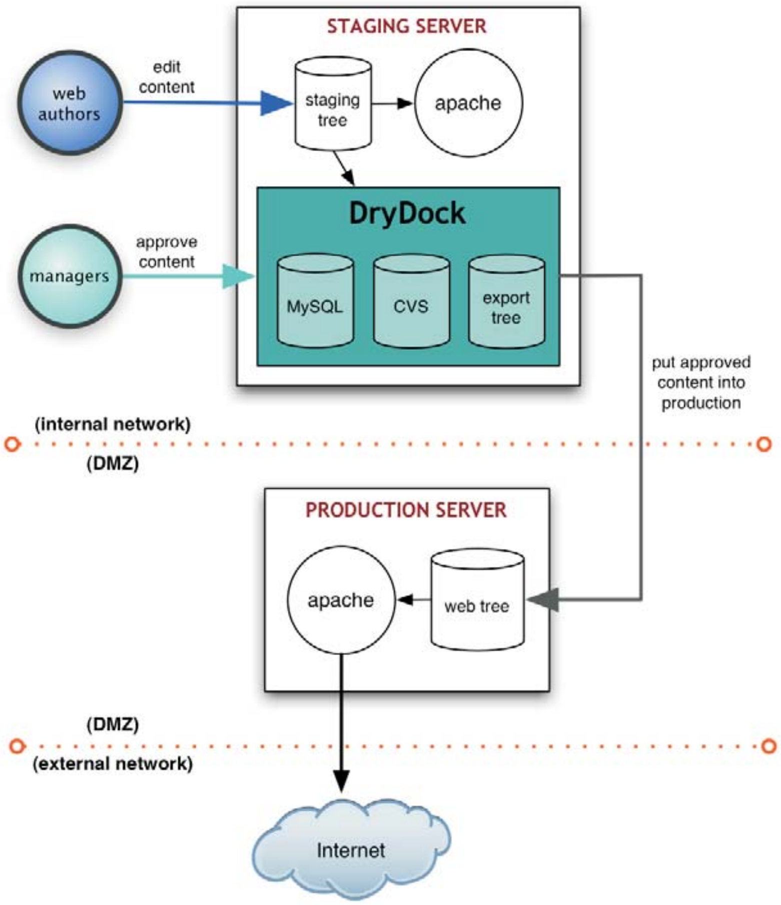


Figure 1: Web publishing with DryDock.

fashioned after Sun's Java Servlet [15] architecture (a paradigm we were familiar with), and includes little extraneous, dynamic componentry we'd have to work around. We could implement all of the heavy-lifting functionality in plain Python and use a small number of servlets to expose a web interface – we found such an architecture much more workable via WebKit than Zope.

Using WebKit, we devised an application that would give us the security and auditing features we required. Several months later, we put DryDock into production. DryDock has been managing our web publishing for over a year and a half now.

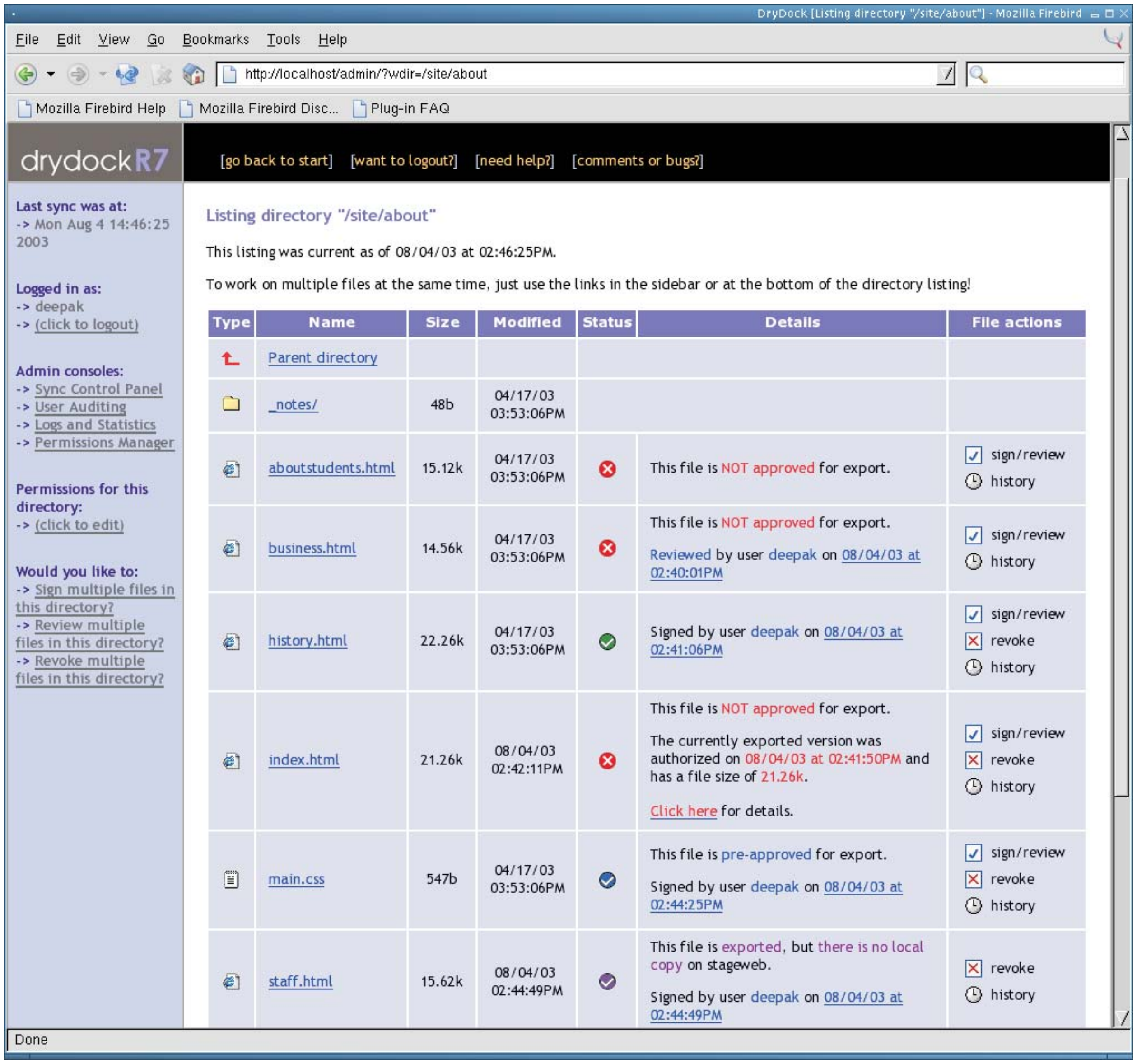
What Is DryDock?

DryDock tackles our information security problem in two main ways: by implementing a dual web server setup, and by forcing a separation between creating content and approving its publication.

Figure 1 details the process of web publishing using DryDock. DryDock's dual web server setup is comprised of a *production* machine and a *staging* machine, both of which have identical web server configurations. The production server holds content suitable for the public and resides in a publicly accessible DMZ[2] [14], while the staging server resides behind the firewall as part of the internal network.

The staging server houses a web tree containing files under development (the *development* tree) and a separate tree containing files authorized for publication (the *export* tree). The development tree is

[2]Demilitarized Zone: a "neutral zone" between a company's private network and the outside public network.



DryDock [Listing directory "/site/about"] - Mozilla Firebird
File Edit View Go Bookmarks Tools Help
http://localhost/admin/?wdir=/site/about
Mozilla Firebird Help
Mozilla Firebird Disc...
Plug-in FAQ
drydock R7
[go back to start] [want to logout?] [need help?] [comments or bugs?]
Done

Last sync was at:
-> Mon Aug 4 14:46:25 2003

Logged in as:
-> deepak
-> (click to logout)

Admin consoles:
-> Sync Control Panel
-> User Auditing
-> Logs and Statistics
-> Permissions Manager

Permissions for this directory:
-> (click to edit)

Would you like to:
-> Sign multiple files in this directory?
-> Review multiple files in this directory?
-> Revoke multiple files in this directory?

Listing directory "/site/about"

This listing was current as of 08/04/03 at 02:46:25PM.

To work on multiple files at the same time, just use the links in the sidebar or at the bottom of the directory listing!

Type	Name	Size	Modified	Status	Details	File actions
	Parent directory					
	_notes/	48b	04/17/03 03:53:06PM			
	aboutstudents.html	15.12k	04/17/03 03:53:06PM	✗	This file is NOT approved for export.	sign/review history
	business.html	14.56k	04/17/03 03:53:06PM	✗	This file is NOT approved for export. Reviewed by user deepak on 08/04/03 at 02:40:01PM	sign/review history
	history.html	22.26k	04/17/03 03:53:06PM	✓	Signed by user deepak on 08/04/03 at 02:41:06PM	sign/review revoke history
	index.html	21.26k	08/04/03 02:42:11PM	✗	This file is NOT approved for export. The currently exported version was authorized on 08/04/03 at 02:41:50PM and has a file size of 21.26k. Click here for details.	sign/review revoke history
	main.css	547b	04/17/03 03:53:06PM	✓	This file is pre-approved for export. Signed by user deepak on 08/04/03 at 02:44:25PM	sign/review revoke history
	staff.html	15.62k	08/04/03 02:44:49PM	✓	This file is exported, but there is no local copy on stageweb. Signed by user deepak on 08/04/03 at 02:44:49PM	revoke history

Figure 2: The DryDock directory listing screen.

accessible to web authors on the internal network, while the export tree is solely managed by DryDock.

Managers (*content approvers*) use DryDock's web interface (Figure 2) to browse the staging server. The web interface presents an integrated and easily discernible view of the development tree and the export tree which shows them:

- which files from the development tree are authorized for publication
- which files still await approval
- the differences between development and export versions of the same file
- any file's approval history

After looking over an unapproved file and finding it fit for public viewing, a content approver *signs* the file, instructing DryDock to mark the file as authorized for publication. For added security, DryDock can be configured to require additional information to complete the signing process, such as the name of the document author or a note explaining why the file is being signed.

Once the file is signed, DryDock automatically copies it from the development tree to the export tree. Finally, DryDock synchronizes the staging server's export tree to the production server over SSH, replacing the external web site with an updated copy containing newly signed files.

Since the staging server's export tree is separate from the development tree, deleting a file from one will not delete it from the other. To remove a file from the export tree, a content approver uses DryDock's web interface to mark that file as *revoked*. This removes the file from the set of files copied to the external web server during the next synchronization.

While signing authority is given to a limited number of users, a larger number can be given *review* authority, or the ability to *soft-sign* files. Reviewing files follows the same process as signing files, save for two important differences: approval information is optional and can be incomplete, and reviewed files still require an authorized signature for publication. At ARL:UT, review is employed by users to partially fill in approval information for a file so that when a signer moves to authorize that file for publication, much of the data required is already present. Review provides much of the same functionality as signing a file, but without a signature's consequences.

Though DryDock's approval process works well for most documents, rapidly changing documents must be continually re-signed by content approvers for each change to appear on the external web site. To ease dealing with these types of files, a user can sign them as *pre-approved*; files marked as such will be copied to the production server during synchronization even if their contents have changed since they were signed. Since file pre-approval circumvents DryDock's typical workflow process, we advise our users to apply the option sparingly; users must be vigilant about pre-approved files' content.

Web Developer Migration

Migrating web developers to DryDock's web publishing process should be painless. Instead of placing content directly onto the production server, web developers now place documents onto the staging server. At ARL:UT, we configured our staging server to support the same access methods our web developers have always used: FTP, Samba, and NFS. Since nearly all off-the-shelf web development tools can use at least one these methods to edit pages, we simply instructed our web authors to re-configure their tools to point to the appropriate directory on the staging server.

In most cases, web developers shouldn't have to change their pages for DryDock. In a proper DryDock setup, the staging server's web server environment mimics that of the production server, so properly functioning pages on the staging server will still work when exported to the production machine.

The lone caveat is the avoidance of absolute links: since the production server (not the staging server) will be serving up content to the public, any links in a web page explicitly mentioning the staging server will not work. Relative links solve this problem by not mentioning the server name in the link address; the web server assumes the document is local.

However, there is one situation in which use of a relative link is impossible. If page authors need to refer to a secure page from an insecure page, they need to refer to the host using https:// instead of http://. Since it isn't possible to specify a protocol in a relative link, an absolute link is required. The solution we employed at ARL:UT was to use a simple server-side-include variable that contains the host name of the machine from which the page is served. Page authors then construct an absolute link, using the variable in place of the host name, and the correct server name is inserted dynamically.

Benefits to Administrators

Improved Information Security

DryDock safeguards against the dissemination of inappropriate web content by codifying a formal document approval process. It ensures that all updates to web content are inspected for propriety *before* they ever escape the shelter of the internal network. DryDock makes users accountable for the documents they approve; if content is found to be inappropriate, administrators can easily determine who let that content through.

DryDock acts as a *document firewall*. Just as a traditional firewall regulates the flow of packets to a private network, DryDock regulates the flow of documents to the production web server; they are both access control mechanisms. Through signing, preapproving, and revoking files, DryDock users create rule sets that manage the movement of documents into and out of its export tree.

Content Auditing

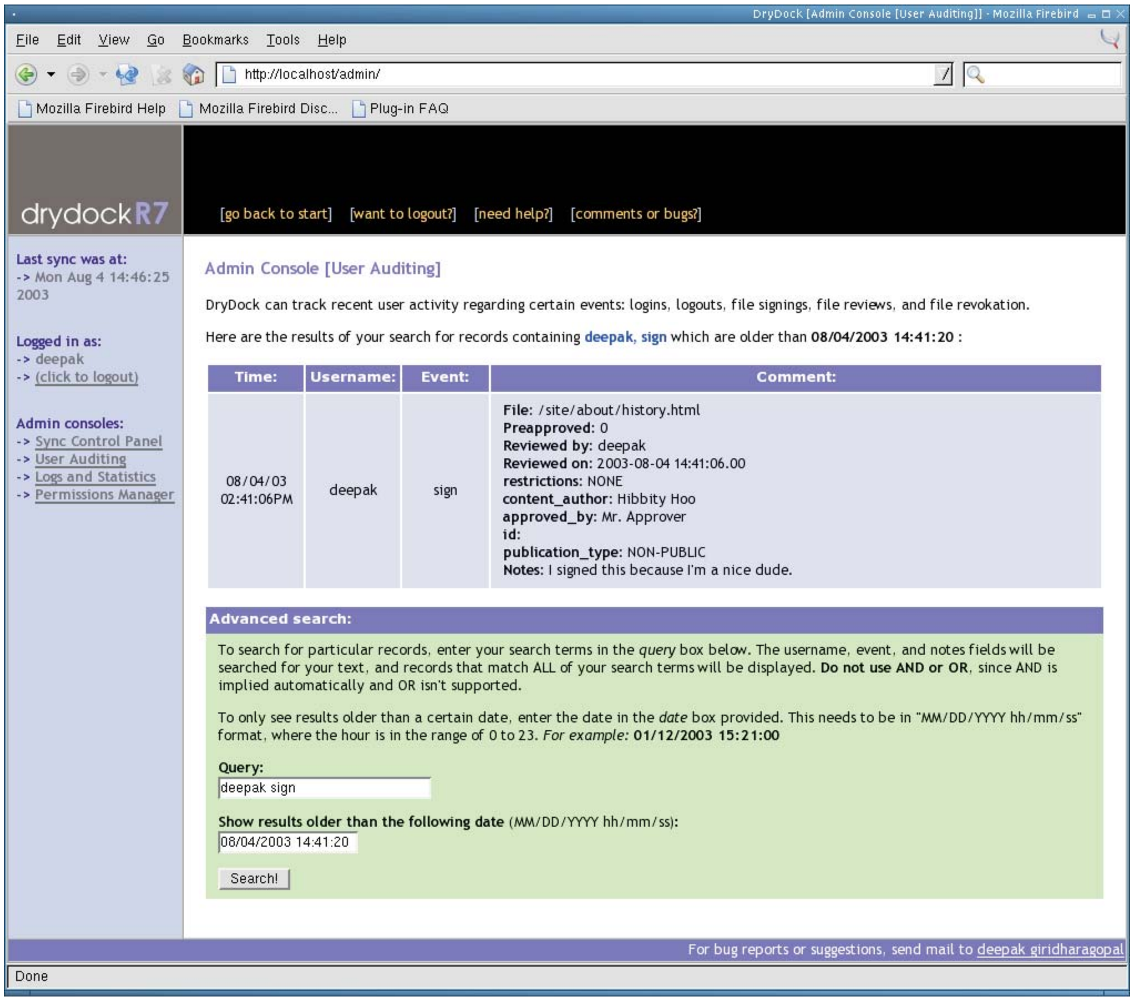
DryDock goes to great lengths to log all user and system activity and keep administrators abreast of all changes in web content. DryDock provides administrators with an auditing console that lets them browse DryDock's records. Administrators can inspect the most recent events, all events pertaining to a user, or any events within a specified date range, or they can perform a free-text search to tailor the results to their liking. For greater detail, administrators can peruse DryDock's extensive log files. To immediately notify administrators of changes in content, DryDock can be configured to send out email whenever activity occurs.

With these tools, administrators can easily diagnose problems with content. If a document is approved that is later found to be unsuitable for the public, administrators can refer to the email DryDock sent when the file was signed; the email indicates the parties responsible for approving the file and the file's full path. Administrators can revoke the document's publication or, if necessary, manually restore a previous version of the file using DryDock's revisioning system. They then can use the auditing console to find any other files upon which the responsible parties have recently operated (Figure 3), and handle them as needed.

Improved Web Server Security

DryDock's dual web server setup makes the production server easy to harden. Users never interact directly with the production machine; DryDock interacts with it on their behalf. This allows administrators to operate the production server in a more secure and regimented network environment than normally feasible. Because normal users never access the web server, system administrators can restrict its logins to a single administrative account. As DryDock updates the server's content over SSH, traditional file access services such as Samba, NFS, or FTP can also be disabled.

If the web site is comprised of mostly static pages and simple scripts, backups are much less


DryDock [Admin Console [User Auditing]] - Mozilla Firebird
File Edit View Go Bookmarks Tools Help
http://localhost/admin/
Mozilla Firebird Help
Mozilla Firebird Disc...
Plug-in FAQ
drydockR7
[go back to start] [want to logout?] [need help?] [comments or bugs?]
Done

Last sync was at:
-> Mon Aug 4 14:46:25 2003

Logged in as:
-> deepak
-> (click to logout)

Admin consoles:
-> Sync Control Panel
-> User Auditing
-> Logs and Statistics
-> Permissions Manager

Admin Console [User Auditing]

DryDock can track recent user activity regarding certain events: logins, logouts, file signings, file reviews, and file revokation.

Here are the results of your search for records containing **deepak, sign** which are older than **08/04/2003 14:41:20** :

Time:	Username:	Event:	Comment:
08/04/03 02:41:06PM	deepak	sign	**File:** /site/about/history.html **Preapproved:** 0 **Reviewed by:** deepak **Reviewed on:** 2003-08-04 14:41:06.00 **restrictions:** NONE **content_author:** Hibbity Hoo **approved_by:** Mr. Approver **id:** **publication_type:** NON-PUBLIC **Notes:** I signed this because I'm a nice dude.

Advanced search:

To search for particular records, enter your search terms in the *query* box below. The username, event, and notes fields will be searched for your text, and records that match ALL of your search terms will be displayed. **Do not use AND or OR**, since AND is implied automatically and OR isn't supported.

To only see results older than a certain date, enter the date in the *date* box provided. This needs to be in "MM/DD/YYYY hh/mm/ss" format, where the hour is in the range of 0 to 23. *For example:* **01/12/2003 15:21:00**

Query:
deepak sign

Show results older than the following date (MM/DD/YYYY hh/mm/ss):
08/04/2003 14:41:20

Search!

For bug reports or suggestions, send mail to deepak giridharagopal

Figure 3: Auditing console.

complicated. Since DryDock rebuilds the entire web tree during each synchronization, there's no need to maintain backups of the production web server beyond a base image of the machine's initial configuration. DryDock simply treats a production web server as a drone – all web content is housed safely inside the firewall on the staging server, and it's pushed out to the external machine when necessary.

Finally, DryDock's repeated rebuilding of the production web tree impedes a naïve intruder from simply defacing web pages. The corrupted documents will simply be reconstituted during the next synchronization.

DryDock Composition

DryDock is written in Python and uses WebKit for request handling and session management. The user interface is written in HTML using Cheetah [2], a Python templating library. DryDock's back-end is comprised of four main components: a relational database that stores auditing information, a role-based permissions system, a revisioning system that tracks changes in approved documents, and a synchronization daemon that updates the production web server.

The Database

Since DryDock needs to query its data on a per-file, per-directory, and per-user basis, storing the information in a relational database was a natural fit. DryDock uses MySQL [9] to store authorization information for files, permission definitions for users, review information, and user activity logs.

Each time a file is signed, reviewed, or revoked, DryDock records the operation's details in its tables. DryDock remembers the user, the time, the file's current MD5 fingerprint, users' notes, and any additional information DryDock has been configured to accept. DryDock uses this data to display a file's transaction history and to determine if a file's contents have changed since it was last signed or reviewed.

Permissions

DryDock features a role-based permissions model. Users and groups from the underlying UNIX system can be assigned a role of *admin*, *sign*, *review*, *view*, or *none* per path on the staging web tree. A role circumscribes all of the actions a user can perform; any capabilities not specifically permitted are prohibited for that directory and all paths underneath. Figure 4 describes the different roles and shows how they are cumulative in design; for example, a user with *sign* authority for a path also has *review* authority.

Permissions for a user resolve in a bottom-up manner. If no role is defined for the user on a path, DryDock searches for one defined for the user on the path's parent directory. The process continues until a role is found or the root directory is reached. If no role is found for the user, then DryDock performs the same recursive check for each group the user is in. If there is still no matching role, DryDock assumes the user has no privileges for the initial path.

The Revisioning System

To let administrators see a document's evolution, DryDock relies on the freely available Concurrent

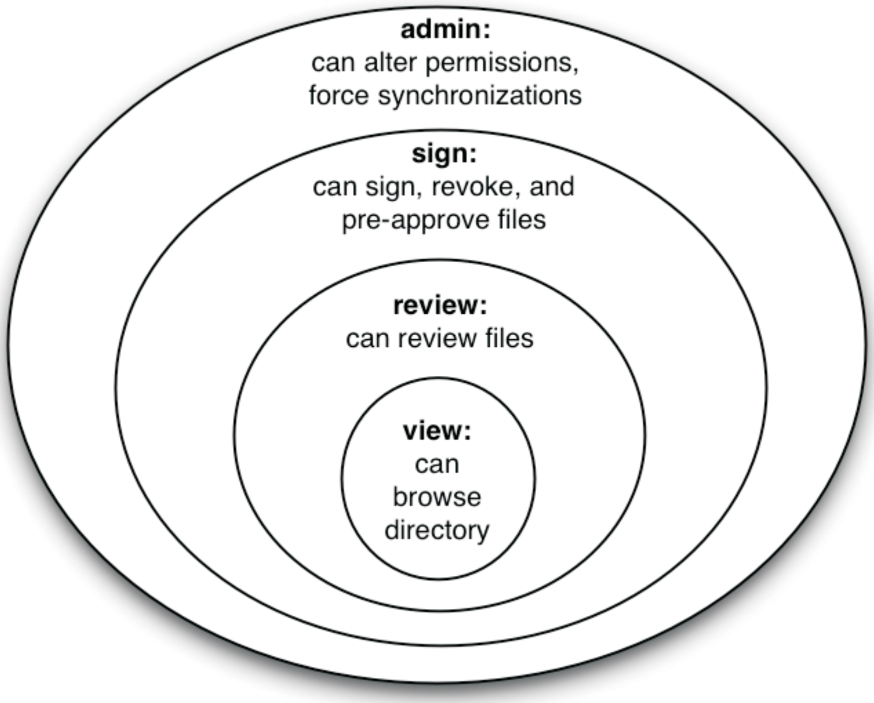

admin:
can alter permissions, force synchronizations
sign:
can sign, revoke, and pre-approve files
review:
can review files
view:
can browse directory

Figure 4: Role-based permissions model.

Versions System [1] (CVS) to track changes in approved files. CVS provides stable, production quality, multi-file version control. DryDock interacts with CVS by coordinating files among three directories during versioning operations: the staging tree, a CVS repository ("a complete copy of all the files and directories which are under version control" [6]), and a CVS working directory (a working copy of the repository used to commit changes to the repository).

When a file is signed, DryDock copies the file to the CVS working directory and then commits it to the repository. Similarly, when a file is revoked, DryDock deletes the file from the CVS working directory and then notifies the repository that the file has been removed. Through use of these mechanisms, the CVS working directory always contains the current version of each approved file.

While CVS adequately handled most of our revisioning needs, its file-based design couldn't handle changes in the repository's directory structure. "Because it uses the RCS storage system under the hood, CVS can only track file contents, not tree structures" [3]. CVS provides no way to remove a directory from the repository without losing all of its versioning information. So, if you delete all the files within a directory and still want to access those files' revision histories, the directory must remain in the repository. This lingering directory prohibits adding a new file with the same name to the repository because UNIX file systems won't allow two identically named items in a directory. If a file cannot be added to the repository, then DryDock users cannot approve that file.

To remedy this, we distort directories' names when they are added to the working directory. Whenever DryDock creates a directory in the CVS working directory, it prepends a predetermined sequence of characters to the directory's name; this mangled name is used when the directory is added to the repository. Concurrently, we prohibit users from creating files beginning with the same reserved characters. This allows us to have directories and files with the same name under version control simultaneously.

Synchronization

With DryDock, users never make updates directly to the production web server. Synchronization (*sync*), DryDock's process of pushing documents out to the production machine's web tree, is the sole way to update the external web site. Periodically, DryDock copies approved versions of all documents to the external machine, reconstructing the production web site.

Since synchronizations are the only way changes propagate to the production machine, we needed to schedule them frequently. Instructing DryDock to immediately sync whenever a user signed or revoked a file would work well in periods of light use, but the staging server would be overwhelmed if users signed and revoked pages en masse.

Our solution was to employ *delayed syncs*. Instead of immediately syncing when a user signs or revokes a file, DryDock schedules a sync to occur five minutes later. If users sign or revoke additional files inside this five-minute window, DryDock reschedules the sync for five minutes from the time of the most recent approval operation. This process continues until there is no activity for the duration of the window, at which time the sync occurs. Since heavy usage would keep pushing the sync back by five minutes, perhaps indefinitely, we instituted a one-hour failsafe between syncs. If a sync hasn't occurred in the last 60 minutes, one is forced. Grouping updates in this way gave us a reasonable compromise between update frequency and server load. For situations requiring finer control, however, we allow DryDock administrators to force a sync on demand.

Figure 5 details the sync process. Sync is split into two parts: handling pre-approved files and exporting signed files to the production machine.

Pre-approved files require special handling during sync to ensure their changes are monitored by

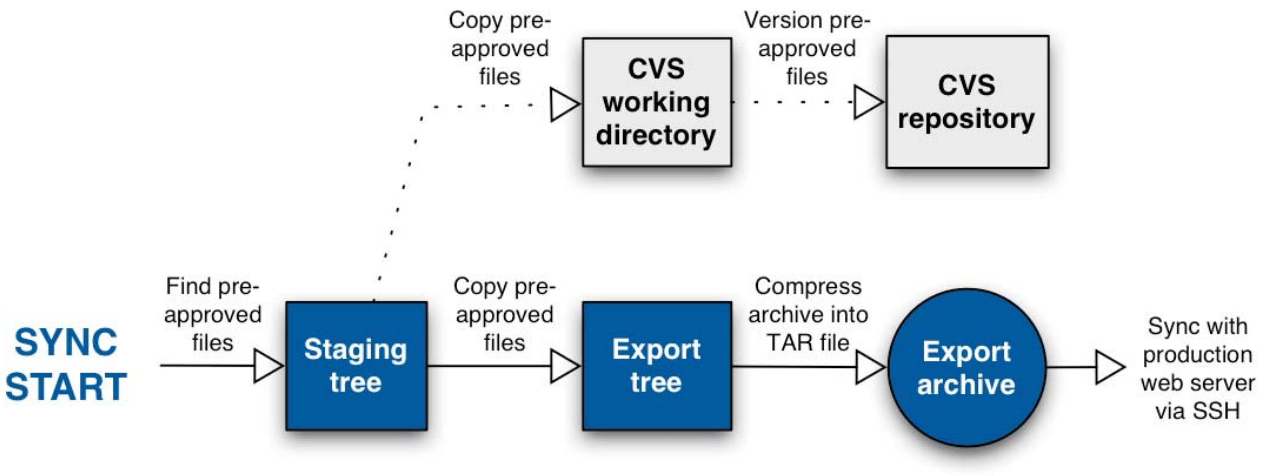


Figure 5: Synchronization flow.

DryDock's revisioning system. Ordinarily, DryDock only adds files to its revisioning system when they are signed or revoked. Since users aren't required to sign pre-approved files each time their contents change, DryDock would normally be unable to track changes in pre-approved documents over time. To remedy this, DryDock adds the current state of every pre-approved file to its revisioning system at the start of each sync.

Before DryDock exports approved files to the external web server, it must construct an image of the production web tree. Originally, we used CVS's export ability. A CVS export would copy the most recent versions of all active files to a temporary directory. Since the CVS repository contains the current versions of all approved documents, this temporary directory would, by definition, contain an image of the production web tree. We then normalized any "mangled" directory names, converted the directory to a tar archive, and copied the file to the external machine.

Action	Time	% of Total
Copy pre-approved files	1s	1.0%
Add pre-appr'd files to CVS	22s	21.7%
CVS export	53s	52.5%
Normalize directory names	2s	2.0%
Tar archival	7s	6.9%
Update the production server	16s	15.8%
Total	101s	

Figure 6: Performance of CVS-export-based synchronization at ARL:UT.

Though this worked correctly, synchronizations performed poorly. After investigation, we discovered that our use of CVS's export facility was a sizable performance bottleneck; the synchronization process spent over half its time waiting for the export to complete (Figure 6).

Our answer was to continually maintain an image of the production web tree in a directory alongside the CVS repository; this is the *export* directory. Whenever a user signs a file DryDock copies it to the export directory, and whenever a file is revoked DryDock deletes it from the export directory.

By continually maintaining this mirror of the production web site, we no longer need to perform a CVS export or normalize any directory names. Instead of creating this mirror at sync time, we can spread the work out over time, updating the mirror as changes occur. Table 6 shows that, by skipping these steps, we cut the sync time by 54.5% from 101 seconds to only 46 seconds.

DryDock can use a variety of user-defined scripts to transmit the tar-archived export directory to the external web server. At ARL:UT, we use SSH. The production server is configured to automatically decompress the archive and replace its current web root with the new content. To make the process more secure, we use a pair of public and private cryptographic keys to establish the connection instead of a traditional user name and password combination, and the key is associated exclusively with the specific script that updates the machine's web root.

Evaluation

All told, DryDock has moved us away from our largely unsecured and unrestricted web publishing process to one that affords us much better information security. It has made it possible for a large number of web authors to safely publish content to a web server that remains almost completely isolated from our internal network. By removing direct user interaction with the external web server, DryDock has allowed us to secure our web server to a degree not before possible [11]. And by enforcing a formal approval process, DryDock has given our management and administrators total publication oversight.

DryDock has been governing our web presence for over one and a half years, and has proven itself to be stable and dependable. This notwithstanding, there are still several areas that could be improved.

User Interface Issues

While DryDock scales well vis-a-vis directories containing thousands of files, its performance is overshadowed by slow web page rendering and the problems users have when operating on such large sets of data. DryDock uses HTML tables to display information about each file, and table rendering is notoriously taxing on web browsers. Consequently, as directories grow massive, a web browser cannot display much of DryDock's user interface quickly.

Furthermore, DryDock's web interface isn't truly designed to operate on large amounts of data. Though users can simultaneously sign, review, or revoke multiple files in the same directory, the usability of a point-and-click, forms-based web interface doesn't scale well when dealing extremely large numbers of inputs.

In spite of these structural issues, we've had few performance-related complaints. Most web developers simply don't work with thousands of files in a single directory – it's far too cumbersome. DryDock, optimized for interactive performance with modest data sets, suits the usage patterns and scale at ARL:UT.

Moving Away From CVS

While we were able to work around many of CVS's directory management problems through name mangling, overcoming its lack of a truly programmable interface is more problematic. To interact with CVS, DryDock is limited to invoking CVS's command-line tools and parsing their output. This output is designed to be easily readable by humans, not easily parsed by software [19]. This hinders DryDock's integration with

CVS, and tight integration with a revisioning system is a prerequisite for DryDock to provide repository management and inspection features.

CVS's directory management issues and its command-line interface are structural problems not likely to be remedied. We will migrate from CVS and towards a revisioning system that better meets our requirements.

Ostensibly, the next-generation version control system that will best suit our needs is Subversion [4, 18]. Subversion tracks changes in directory structure, obviating our name mangling. More importantly, however, Subversion has a well-defined C programming interface and a nascent Python interface (DryDock's native tongue). Subversion is currently in development and we await its first stable release.

File Pass-through

Since DryDock tracks changes in all approved documents, the size of DryDock's CVS repository increases as time goes on. Pre-approved files exacerbate the problem since their changes are committed to CVS at each sync; as syncs occur at least once per hour, pre-approved files are committed to CVS at least 24 times daily.

One solution to this problem is to allow users to flag signed files as *pass-through*: files marked as such will not be added to DryDock's revisioning system. A signer marks a file as pass-through if tracking its changes over time isn't important. Marking pre-approved files as pass-through can slow the repository's growth, though at the expense of thoroughness.

Incremental syncs

To reduce sync time, DryDock could transmit updates to the production machine during each synchronization instead of transmitting the entire web tree. During a sync, DryDock could query the database to determine which files have been signed or revoked since the last sync. It could use this information to construct an archive containing any recently signed files, all pre-approved files, and a text file listing which files need to be deleted from the production server. This archive could then be transmitted to an update script on the external machine that will place the updated files in the correct locations and delete any files specified in the deletion list.

For added safety, these incremental syncs can be intermingled with full syncs. For added performance with truly large web sites, DryDock could even employ a binary differences algorithm such as XDelta [21] to transfer only the parts of individual files that have changed.

Initially, it would seem appropriate to use rsync[3] [12] to reconcile and transmit the differences between the export directory and the production server's web tree. However, rsync is designed to operate over high-latency, low bandwidth links; at ARL:UT, we have a switched 100-megabit connection between the staging and production servers. Furthermore, unlike rsync, DryDock is omniscient; it knows before opening a connection to the production server which files require updating. Instead of spending connection time determining which file chunks need to be sent over, DryDock can simply transfer data.

[3]An incremental file transfer tool. "rsync uses the *rsync algorithm* [21] which provides a very fast method for bringing remote files into sync. It does this by sending just the differences in the files across the link, without requiring that both sets of files are present at one of the ends of the link beforehand." [13]

Availability

DryDock is currently available at http://www.arlut.utexas.edu/DryDock, and we plan on having placed DryDock under the GNU Public License by the time of the conference.

DryDock requires Python, Webware, CVS, SSH, and MySQL. DryDock has been designed and tested on Linux and Solaris, and it is expected to run on any modern UNIX platform that supports the aforementioned tools.

Acknowledgments

Many people have contributed to DryDock's development. Jonathan Abbey greatly assisted in DryDock's design and made important contributions to DryDock's permissions handling and synchronization routines. Dan Scott lent administrative support to the project and helped lay out many of DryDock's initial requirements, and Nanette Lemma provided invaluable user feedback.

I'd like to thank Jonathan Abbey (again) and Chad Duffy for tirelessly reading and commenting on the many iterations of this paper. I'd also like to thank Michael Gilfix, this paper's steward, for his refinements.

Lastly, I'd like to thank the authors of Webware for their wonderful work, the developers of Python for a great programming language, and Applied Research Laboratories for supporting my software development efforts, both large and small.

What's in a Name?

The name "DryDock" is derived from nautical terminology. A drydock is "a specialized dock where boats are pulled out of the water to be repaired, painted, or inspected" [5]. We found the analogy appropriate.

Author Information

Deepak Giridharagopal is the primary developer of DryDock. He holds a B.S. in computer science from The University of Texas at Austin. After working for Reactivity, Inc. doing Enterprise Java development, he now works at Applied Research Laboratories writing software for their Computer Science Division. Besides all things computer-related, he is thoroughly entertained by automobiles, The Smiths, giant transforming robots, and breakdancing. He can be reached by email at deepak@arlut.utexas.edu.

References

[1] Berliner, Brian, "CVS II: Parallelizing Software Development," *Proceedings of the USENIX Winter 1990 Technical Conference*, USENIX, pp. 341-352, 1990.

[2] *Cheetah – the python powered template engine*, http://www.cheetahtemplate.org , 2003.

[3] Collins-Sussman, Ben, "The Subversion Project: Building a Better CVS," *Linux Journal*, Num. 94, 2002.

[4] Collins-Sussman, Ben, Brian Fitzpatrick, and C. Michael Pilato, *Subversion: The Definitive Guide*, http://svnbook.red-bean.com/html-chunk/, 2003.

[5] *sitesalive Glossary*, http://www.sitesalive.com/admin/glossary/sectD.html , July, 2003.

[6] Fogel, Karl Franz, *Open Source Development with CVS*, The Coriolis Group, 1999.

[7] Haakenson, Vanessa, *What is postnuke?*, http://noc.postnuke.com/docman/view.php/5/12/whatispostnuke.htm, June, 2003.

[8] McKay, Andy and Amr Malik, *The Plone Book*, http://www.plone.org/documentation/book , June, 2003.

[9] *Mysql: An Open Source Relational Database*, http://www.mysql.org , 2003,

[10] *O'Reilly network: Wiki Wiki Web*, http://www.oreillynet.com/pub/d/282 , 2003,

[11] Rhodes, Charles, "The Internal Threat to Security or Users Can Really Mess Things Up," *GSEC Practical*, http://www.sans.org/rr/papers/8/856.pdf, 2003.

[12] *Rsync*, http://rsync.samba.org , June, 2003.

[13] *Rsync Readme*, http://rsync.samba.org/README.html , 2003.

[14] Runnels, G. Michael, "Implementing Defense in Depth at the University Level," *GSEC Practical*, Num. 1.4, http://www.sans.org/rr/paper.php?id=596 , 2002.

[15] *Java Servlet Technology*, http://java.sun.com/products/servlet/ , July, 2003,

[16] Spicklemire, Steve, Kevin Friedly, Jerry Spicklemire, and Kim Brand, "Zope: Web Application Development and Content Management," *Que*, 2001.

[17] *Vignette – Content Management and Portal Solutions*, http://www.vignette.com , July, 2003

[18] *Subversion: A Compelling Replacement for CVS* http://subversion.tigris.org , June, 2003.

[19] Taler, Alexander, *libCVS*, http://libcvs.cvshome.org/servlets/ProjectHome , June, 2003.

[20] *Tikiwiki*, http://tikiwiki.sourceforge.net , 2003.

[21] Tridgell, Andrew and Paul Mackerras, "The rsync Algorithm," *Tech. Report TR-CS-96-05*, Australian National University, June, 1996.

[22] *Webware for python*, http://webware.sourceforge.net , 2003.