Cro-Magnon: A Patch Hunter-Gatherer

Jeremy Bargen – University of Colorado at Boulder and Raytheon Systems Company
Seth Taplin – University of Colorado at Boulder and CiTR, Inc.

ABSTRACT

On a relatively large and heterogeneous network, there may be several operating systems and dozens of major applications in general use. Locating and maintaining patches for these systems can take up a significant portion of a system administrator's time. In addition, groups of machines must all be kept at consistent patch levels, and the exact patch level may vary depending on the group. Security patches are especially problematic because they appear at irregular intervals, and the administrator generally wants to find and install them as soon as possible after they become available.

This paper describes Cro-Magnon, a system for automating the process of patch downloading and application. Cro-Magnon can be configured with a list of patch sites and will mirror those sites, downloading new patches as they are detected and notifying the administrator of the downloads. Cro-Magnon can verify patch authenticity and can maintain patch data for multiple machine groups and architectures, all with different administrators.

The Cro-Magnon architecture is intended to be as flexible as possible. It allows for multiple download methods such as FTP and HTTP and multiple authentication schemes like MD5 and PGP. Although it currently deals primarily with patch downloading and notification, it is intended to be extended to allow automated patch application and maintenance.

Introduction

System administrators must often maintain large networks of heterogeneous systems. In order to keep the network as secure as possible, it is important that the system administrator be aware of the latest versions of all security patches as soon as they become available. In addition, administrators must often keep operating systems as well as applications up-to-date with the latest bug fixes. In some cases (especially for software development environments), the administrator must even maintain all machines in one logical group at one consistent patch level, while machines in another group are maintained at a different level.

For a large network, the amount of work involved in patch management can quickly become overwhelming. System administrators may not have time to locate and download security patches until days or weeks after a security advisory is first issued. This can cause periods of dangerous insecurity for the network. In addition, when the network is composed of multiple types of operating systems, each with several patch sites, it becomes easy to miss some sites altogether, so that some fixes are never installed at all.

Cro-Magnon is a system that automates the process of patch gathering in two ways: first, it will search a specified list of Internet sites for patches and automatically download, verify, and notify the administrator about the patches. This enables the administrator to find out about new security and bugfix updates promptly and reduces the chance that any one site will be overlooked. Second, Cro-Magnon is ultimately intended to simplify the process of applying patches and distributing them to select groups of machines. This paper describes a full (although simple) implementation of the first goal.

Functional Overview

Cro-Magnon has two distinct modes of operation. The primary mode is as a daemon, for periodic background checking and notification of new patch status. Cro-Magnon can also run in an interactive mode, giving a "users-eye" view into the current state of the configured systems and applications and their available patches. Downloaded patches are stored in a hierarchical layout in the filesystem, separated by system architecture and applications.

In daemon mode, Cro-Magnon detaches from the shell and runs in the background, periodically waking up and checking to make sure that its patch database is up to date with the latest patches available for its configured systems and applications. If a new patch is found, it is automatically downloaded into the filesystem, optionally verified via MD5 or PGP signature file (as defined by the configuration) and the system administrator is notified (typically by email) of the patch's arrival. This mode allows a system administrator to start Cro-Magnon from a startup script and let Cro-Magnon automate the generally routine job of checking vendors and distribution sites for the latest patches.

Interactively, Cro-Magnon provides a simple console-based user interface so that a system administrator can examine the current state of the configured systems. The user interface provides the rudimentary ability to view the known systems, machine groups, and applications that are currently monitored, and the various patch levels that Cro-Magnon has detected for them.

System Architecture

The Cro-Magnon system is composed of a core ‘‘Engine’’ module, written in Perl, surrounded by various plug-in Perl modules (Figure 1). Most of the functionality of the system is supplied by the plug-ins, including patch downloading, patch verification, administrator notification, and the user interface. The goal of this design is to allow customized behaviors for any or all of the basic functions of the system, simply by writing modules that follow a documented API. The only functions in the Engine are those which wire the plug-ins together using the configuration file and the main event loop for the daemon mode.

The configuration file divides the network up into ‘‘Systems,’’ which are logically distinct hardware and/or software architectures; ‘‘Applications,’’ each of which may represent a single application, an application suite, or the operating system itself; ‘‘Sites,’’ which are locations such as FTP and Web sites from which patches may be downloaded; and ‘‘Groups,’’ which associate a list of machine host names with a list of Applications, each of which may have a version number associated with it, so that different versions can be tracked for different groups or systems.

Each Application is associated in the config file with a number of Sites, each of which is associated with both a download module that tells it how to retrieve patches from that Site and a verification module that tells it how to verify the patches once they have been downloaded. Examples of possible download module functionality include anonymous FTP, FTP using a certain ‘‘registered user’’ account, HTTP, or even copying files from another directory on the same system. The most general approach is to retrieve every file in a directory, however, modules can be specialized to download specific files or filetypes as each situation requires. Examples of verification modules are MD5 or PGP verification, or ‘‘Simple’’ (no verification).

All patches are stored in a hierarchically structured section of the filesystem. The patch root directory is defined in the configuration file; one subdirectory exists under this root for each defined System. In a System’s directory, each Application has a subdirectory. This subdirectory in turn contains one directory for each Site containing Application patches. This allows patches to be stored indefinitely for each remote site with no danger of overlap.

When running in daemon mode, Cro-Magnon will run in the background and will periodically wake up and download updates from the Sites it knows about. The system may be run as a daemon, in which case it will wake up periodically after an interval specified on the command line; or it may be told to run only once via a command-line flag. In the latter case, it is assumed that a tool like cron will be used to run Cro-Magnon periodically. This approach gives the end user maximum flexibility as to how the system is run. The download strategy is defined by the individual download module but should generally follow a ‘‘FTP mirror’’ strategy: a file is downloaded if it is new or if it has changed since the previous download of that file. If any files are downloaded from a specific Site, they will be verified using the Site’s verification module. A download report listing the files downloaded, the Site they were downloaded from, and their new location in the patch database will then be generated and sent to the Group administrators interested in those files.

In interactive mode, Cro-Magnon functions as a patch viewer. The user can specify the exact System,

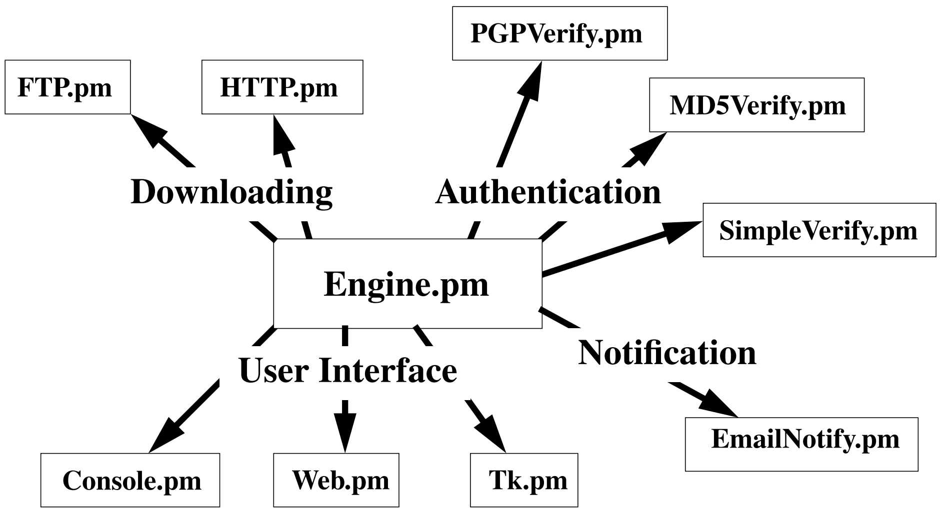

Figure 1: The Engine and sample plug-ins

Group, and Application of interest, and may then see all the downloaded patches for the application and Cro-Magnon's best guess as to which of them are installed. This 'best guess' is dependent on the ability of the software to determine the current version of a given application that has been installed, directly through Cro-Magnon, or possibly from additional information provided by the operating system for "manual" updates. Since there is little consistency in version identification across applications and operating systems, a version module must be used to return the current version of an application. For example, the Linux kernel version can usually be found using 'uname -rv' while perl and gcc both support a '-v' flag. In addition, interactive mode serves as a framework for many of the desired future extensions to the system (see "Future Enhancements").

Software Architecture

For any system administrator who has had to deal with a heterogeneous network, a primary concern for a "universal" tool is portability. To maximize the portability and extensibility of the Cro-Magnon system, it has been designed in the Perl language using extensible modules that provide a skeletal API framework. This framework, coupled with a flexible configuration file format, allows implementation modules to be specialized for unique environments.

The config file is built up from hierarchical keywords which define various records that describe the user's system, groups and applications. At the highest level, the "System <system name>"/"EndSystem" keywords delimit a system description, and "DefaultSystem <system name>", "DefaultGroup <group name>" and "DefaultApplication <application name>" keywords specify which system, group, and application (if any) will be selected as the defaults for interactive behavior.

An Application record is delineated by the "Application <application name>"/"EndApplication", and specifies a number of sites to check for

```
System   TestSys
   Application   TestApp1
      Site TestSite1
        Location   ftp://ftp.h2net.net/pub/nvb/test
        Location   ftp://ftp.somewhere.com/pub/mirrors/h2net/nvb/test
        ToDownload FTP.pm
        ToVerify SimpleVerify.pm
     EndSite

      Site TestSite2
        Location   ftp://ftp.freebsd.org/pub/FreeBSD/CERT/patches/SA-97:06
        ToDownload FTP.pm
        ToVerify SimpleVerify.pm
      EndSite
   EndApplication

   Application   TestApp2
      Site TestSite3
        Location   ftp://metalab.unc.edu/pub/gnu/grep
        ToDownload FTP.pm
        ToVerify SimpleVerify.pm
      EndSite
   EndApplication

   Group TestGroup1
      Machines
         saclass saclass-5
      EndMachines
      CurrentVersions
           TestApp1 Version-v saclass
           TestApp2 Version-RCS saclass
      EndVersions
      Administrator jbargen@acm.org
      Administrator taplin@cs.colorado.edu
      ToNotify EmailNotify.pm
   EndGroup
EndSystem

DefaultSystem TestSys
DefaultGroup TestApp1
DefaultApplication TestGroup1
```

Figure 2: Sample Config File

patches. Each site record is offset by a pair of "Site <site name>"/"EndSite" tags, and is composed of one or more locations designated by a "Location <URL>" entry (multiple locations designate multiple mirror sites), the module to be applied for the download mechanism "ToDownload <module name>", and the module to be applied for verification of the trustworthiness of a patch (for example MD5 checksum verification) denoted by the keyword pattern "ToVerify <module name>".

A Group record is bounded by a pair of "Group <group name>"/"EndGroup" keywords, and contains a machine list between a set of "Machines"/"EndMachines" tags, one or more "Administrator <email address>" keyword pairs, and a list of applications and versions that are currently installed. This application list is formatted between "CurrentVersions"/ "EndVersions" tags, with an application name followed by the name of (and arguments to) a module which will return the current version. The version information is intended to be used to determine if a newly-found patch is of interest to the application, narrowing the selection criteria for notifying the appropriate administrators. The type of notification is specified by a ToNotify keyword.

The Engine can run autonomously as a daemon, checking sites for new patches, or interacting with a specified Display component, for example, a Console display. Display interaction is done through a callback mechanism, to allow for the development of graphical interfaces in the future. This opens up the possibility of specialized Displays that interact with the Engine through a Perl/Tk GUI, or even a Java GUI, which could be embedded in a web page for remote administration over a secure network connection.

Major Software Components

Each of the components of the CroMagnon system is composed of one or more Perl modules. All of the modules except for the Engine may be overridden by derived modules.

Engine

The Engine module is a stand-alone module providing the core functionality of the system. It is the only module that may not be overridden by a derived class. It is the glue that binds the rest of the modules together via the config file. The Engine's primary functions are parsing the config file to understand the Systems, Groups, and Applications that are specified, knowing (from the configuration) which modules to delegate to when dealing with each of these specifications, and whether to run in daemon mode to find and gather patches, or to interact with a user.

The initialization and reading of the config file is common to both operational modes, but the behavior of the Engine changes dramatically depending on the mode in which it is run. Running in daemon mode causes the Engine to detach from the console and periodically wake up and check for new patches. It processes each System in turn, finding and downloading new patches. The Engine then examines each Application's new patches and determines if there is a Group that is interested in that patch. If so, the Engine notifies that Group's administrators that the patch has been downloaded, verified, and is available for installation from a given directory. Sending a SIGHUP to the daemon causes it to re-read and re-parse the configuration file to pick up any recent changes, and then continue on its merry way.

In interactive mode, the Engine interacts with a Display component, providing configuration and patch information as described in the following section.

Display

The display component provides a user interface to the Cro-Magnon system. Although there is not much a user can currently do interactively, this serves as a placeholder for the functionality described later under "Future Work." The only user interface that is currently provided is a simple command-line display, but the architecture will support any interface that can be set up to use an "event loop" style of operation, including a conventional GUI using a package such as Perl/Tk, or even a separate process communicating with the Engine via a socket stream.

The display component interacts with the Engine by use of a dual-callback mechanism. The display runs an event loop which waits for user input. The user enters a request, which causes the display component to call a function in the Engine. The Engine does whatever processing is required, and invokes a callback function in the display that does whatever is required to inform the user of the Engine's actions. When the callback function returns, the display component goes back into its event loop.

In order to create a new display module, an implementor must override the Display abstract base class. There are a number of functions which must be overridden in this class, most of which are callbacks from the Engine. The display architecture does not specify anything about its interaction with the user; this is left entirely up to the display implementation. This allows concepts such as a 'display' component whose 'user' is actually a separate process.

Download

The download component is responsible for retrieving patch files and their verification signatures and storing them in the filesystem. The only download component currently supplied with the system is a simple anonymous FTP mirroring system that downloads everything it finds in a given directory. However, the interface supports practically any kind of download that can be represented by a standard URL for new means of downloading files, and new modules can be derived which are more selective about which files are downloaded. For example, a "Solaris-PatchFTP" module could be implemented that knows exactly which files on a Sun FTP site are applicable Solaris patches, and only retrieves those files. The generic FTP module puts more power and responsibility in the hands of the user, since the user has to decide which files have meaning and deal with them appropriately. Because this "sledgehammer" approach is not always desirable, the download mechanism can be easily refined and extended to make Cro-Magnon a more delicate tool.

In order to create a new download component, an implementor needs to provide only two functions: a constructor, which takes a URL and does any required parsing and initialization, and a download method which retrieves the file. The basic FTP module breaks this method up into several submethods, any of which can be individually overridden. This allows for extensions as simple as recursive mirroring of subdirectories or usage of an FTP site which requires a customer login, or as complex as transferring files via HTTP or from a CD-ROM.

Verification

The verification component is responsible for performing security authentication of the trustworthiness of a downloaded file. Two examples are a "Simple" verifier and a MD5 checksum verifier. The simple verifier just trusts all files to be secure. The primary purpose of this module is to serve as a place holder and define the expected API for future authentication schemes, but it also works for a patch site that is implicitly trusted. The MD5 checksum verifier calculates the checksum of the downloaded patch, and compares it against a downloaded checksum before giving approval for the file. The verification interface is generic enough that it can support many other verification schemes, including PGP signature verification.

To implement a new verification scheme, the module requires only two functions: a constructor which does any required initialization, and a verify method, which takes an array of filenames to be checked, and returns an array which is the subset of filenames which have passed verification. This allows the user to pass in any required files, such as the patch file and the MD5 checksum (or PGP signatures), use them as needed to perform the verification, and return the patches which have passed.

Notification

The notification component takes a list of administrator IDs and a list of downloaded patches and notifies those administrators that the patches have been received. The currently supplied notification module does email notification and assumes that the administrator IDs are email addresses. However, since the ID is free-form text, it may represent anything relevant to the appropriate module: for example, if a notification module is used that writes a message to the administrators' workstation consoles, the administrator IDs could be machine names.

Related Work

Although we presume that most large networks use homegrown automated tools to provide many of the functions included in Cro-Magnon, little has been published about such systems. One goal of the Cro-Magnon project is to provide a freely-available alternative to reinventing the wheel at every new site.

One of the most commonly-used such solutions is some variation on one of the widely-available "mirror" packages. These systems mirror various FTP sites to local directories as Cro-Magnon does, but have no built-in capability for patch verification or notification of administrators. In addition, many vendors have convoluted FTP structures that cannot be handled by simply mirroring a directory. Cro-Magnon modules can be written to deal with these sites, but no easy alternative exists with mirror packages.

Another solution that is fairly easily implemented is a combination of cron and the GNU wget package, which can recursively mirror Web and FTP sites. Wget offers many benefits over a simple mirroring strategy including the ability to do wildcard matching on filenames and good handling of slow or unstable connections. However, the cron/wget combination does not automatically support patch verification or administrator notification. Some sort of additional functionality must be added to provide these capabilities. Wget's strengths could be utilized by Cro-Magnon by creating a download module that calls wget.

A more complete home-grown system is the SAGE-AU (System Administrators Guild of Australia) FTP site (ftp://ftp.sage-au.org.au/), which performs local mirroring of system patches. This system was developed (and is only used) in-house by SAGE-AU. SAGE-AU uses a publicly available 'mirror' Perl script to handle the mirroring of patches to a central location. This script is launched multiple times in parallel, one process for each site, to prevent efficiency bottlenecks and to prevent crashing the whole system if a single connection is hung or broken. Currently, a SQL database is created from the output of the mirroring script. Another process compares this database with the registered interests of each user of the system, and email is generated to notify the appropriate users.

Because the purpose of this system is to mirror available patches, rather than directly gather the patches for installation by an end user, there is no need for the SAGE-AU system to authenticate the patches it mirrors. The end user will want to authenticate these patches, just as they would from any other internet site (even if the SAGE-AU administrators are the end users). Even though the purpose of the SAGE-AU system differs from that of Cro-Magnon, Cro-Magnon's robustness and efficiency could be improved by studying the design of the SAGE-AU system. The most immediately applicable improvement is the parallelization of mirroring sites, but some useful insights might also be gained by studying the mirror script and database at the heart of SAGE-AU.

Though built to serve quite a different purpose than Cro-Magnon, the system monitoring tool called Pulsar [4] uses a somewhat similar architecture to provide extensibility and flexibility. Pulsar, written in Tcl/Tk, periodically monitors the health of various components of a computer system using modules specified in a configuration file and reports problems to the user by means of a configurable ''pulse monitor.'' The monitors that are used, the frequency with which they are activated, and the definition of what constitutes an ''alarm'' are all configured independently by the user.

This design is very similar to Cro-Magnon's in that it provides a great deal of flexibility through the use of pulse monitors which interact with the scheduler and display using a simple API. Like Cro-Magnon's modules, Pulsar's pulse monitors may easily be extended by end users to provide any degree of functionality desired. It is interesting to see how a similar design can be independently applied to two systems with such dissimilar purposes. Further examination of Pulsar's design is warranted.

Future Enhancements

Cro-Magnon has a long list of desired future enhancements; these include:

- Improve system speed and robustness
  - Split the monolithic config file into several smaller files
  - Determine patch dependencies, so that any other required patches are also downloaded, and their dependencies noted when notifying the administrator
  - Bullet-proof the system as much as possible, to avoid single points of failure
  - Parallelization of site access
  - Deal with the situation in which a vendor revokes a patch that was previously available
- More modules
  - FTP user/password authorization
  - HTTP transfers
  - Copies from another directory on the same network
  - Secure transfers via scp/sftp
  - PGP signature verification
- Display improvements
  - Perl/Tk or Java GUI
  - Web-based interface
  - Retrieval and viewing of patch documentation
- Automated patch application
  - Determination of whether a given patch is already installed
  - Patch chaining – cannot install patch y until patch x is installed
- Automated patch distribution
  - Possibly using rdist, or a 'reverse rdist' type of system

Obtaining the Software

Cro-Magnon is still considered pre-alpha release software, and as such is not yet widely available. Please contact the authors for the current software location.

Cro-Magnon requires Perl and several of its modules in order to run. A full listing of requirements is included in the software distribution. All modules may be downloaded from CPAN at www.cpan.org. Minimal requirements are:

- Perl 5.004 or better
- The Sys::Syslog module (standard with the Perl 5.004 distribution)
- The Digest::MD5 module (for MD5 verification)
- The libnet module package (for anonymous FTP downloads)

Acknowledgments

Dan Farmer was helpful in critiquing our design and ideas about the system. He has had many helpful suggestions, most of which we have not yet had time to implement. His suggestions helped us to stay focused on the useful functionality of the system rather than on bells and whistles.

David Conran was willing to share information about the design and inner workings of the SAGE-AU ftp site, which gave us several good ideas for improving Cro-Magnon with respect to efficiency and speed.

The authors of the Net::FTP module (Graham Barr) and the Digest::MD5 module (Gisle Aas) have saved us a lot of work, and we are forever in their debt.

The Perl Cookbook by O'Reilly and Associates had many solutions to problems we encountered in our implementation, including how to detach from the shell for daemon mode and a concise way to parse config file lines.

Our LISA shepherd, Adam Moskowitz, had many useful suggestions for improving the quality of this paper, as did the anonymous referees who critiqued this paper. We are grateful for their help.

And finally, we would like to thank Evi Nemeth, who taught the graduate course in Systems Administration at the University of Colorado, where all this began, and who encouraged us to submit our work to LISA.

Author Information

Jeremy Bargen recently completed his graduate studies at the University of Colorado, Boulder, with a

MS-CS. He currently works for Raytheon Systems Company in Aurora, Colorado, where he is a Senior Software Engineer in the Mission Management Systems organization. Reach Jeremy via U.S. Mail at Raytheon Systems Company; Bldg. S75, M/S A2707; 16800 East Centretech Parkway; Aurora, CO 80011. Or reach him electronically at jbargen@acm.org.

Seth Taplin is pursuing part-time studies for a MS-CS at the University of Colorado, Boulder. He currently works for CiTR, Inc. in Boulder, Colorado, where he is a Senior Software Engineer providing custom software solutions for a variety of domains. Reach Seth via U.S. Mail at CiTR, Inc; 1200 28th Street; Suite 305; Boulder, CO 80303. Or reach him electronically at staplin@acm.org.

References

[1] Dan Farmer, personal communications, April 1999.

[2] *Perl Cookbook*, O'Reilly and Associates.

[3] CPAN documentation for Perl modules, http://www.cpan.org .

[4] R. A. Finkel, "Pulsar: An Extensible Tool for Monitoring Large Unix Sites," *Software Practice and Experience*, Vol. 27(10), 1163-1176.

[5] David Conran, personal communications, August 1999.

[6] R. L. Rivest, *RFC 1321: The MD5 Message-Digest Algorithm*, April 1992.

[7] M.I.T. PGP Distribution, http://web.mit.edu/network/pgp.html .