Cluster Processing for Password Audits

Joseph Kaufeld
Ball State University
jpkaufeld@bsu.edu

ABSTRACT

Password auditing programs are useful for security assessments in large companies and for understanding password strength and complexity. In the past decade, many software applications have been upgraded to cluster-based applications for speed and stability, but password audits, which are traditionally very computationally intensive, have not made the jump to clusters. The author originally wrote his own password auditing software as a proof of a concept called Password_Buster, and this paper details the author’s ongoing work in converting the original password auditing program to a new entity, Cluster_Buster, designed to run on a Beowulf cluster. Steps for upgrading the software as well as current limitations are discussed. Future directions for research are suggested.

Keywords:
Passwords, auditing, Beowulf cluster, Python, OpenSSL

INTRODUCTION

The protection of sensitive information is of the utmost concern for businesses. In order to create a secure network, it is important that every password is strong enough to resist hacking.

The need for secure networks in the business world inspired information security professionals to create password auditing programs. Password auditing programs attempt to break password hashes (a one-way encryption of a string; in this case, the password) (Johansson, 2009). The ease with which these hashes are broken determines the strength of each password, and gives network administrators the types of hashes that were broken (e.g., short passwords or all lowercase characters). This information allows the administrator to assess problem areas and recommend new password requirements.

Typically, password auditing programs to date operate on a single computer. Unfortunately, this causes the software to be limited to whatever resources that particular machine has at hand and forces the usage of custom-built, single purpose machines. Programs such as Ophcrack, Cain and Abel, and L0phtcrack are single-system programs. Password auditing programs are rarely run on a cluster system. Such a system allows more computers to work together to break the same set of hashes; this type of system is much faster, but these types of systems are often found in shadowy enterprises operating outside most laws and are very expensive. Cluster-based programs can be scaled up, meaning more computers can be added when necessary. However, cluster-based password auditing programs are not commonly available to the general public. The author is attempting to fill this void by building the open source Cluster_Buster (C_B) software to run on a Beowulf cluster and make it available to the general public.

The author took his original software, Password_Buster, and modified it to run on new equipment, renaming it Cluster_Buster in the process. This paper describes the procedure and results of the author’s work to convert the original software from a single-resource application to a cluster-based option for password auditing. C_B and its predecessor are programs written in the high-level Python language, and C_B can be operated on any cluster of machines located on the same network. It was made for the use of anyone, from network administrators to hobbyists, and it can run three types of attacks: brute force, dictionary, and hybrid.

METHODS

Cluster_Buster is a fully functional command line program that features three types of attacks: brute force, dictionary, and a hybrid of the two. It gets the hashes by importing a *.pwdump file, a standard output from programs like SAMInside and PWDUMP2 (written by Todd Sabin). Once imported, the program allows a choice of the three types of attacks to leverage against the contents of the *.pwdump file.

The brute force attack is the simplest attack of the three – it starts at “a,” encrypts it, checks the hash it just generated against the one that it imported earlier and looks to see if they match. If they don’t, it tries again with “b,” and so on; “aa” becomes
“aaa,” etc. The brute force attack is also the most flexible – it allows you to create a custom character set before the attack runs, choosing from lowercase ASCII characters, uppercase characters, numbers, symbols, and homoglyphs (characters that look like standard ASCII characters, but are not, are usually obtained through the use of ALT codes, like ALT+141). Once the character set is built, it runs through each item in the set recursively until it either matches the password or hits the maximum length. After a hash is loaded, the program begins creating a list of all the possible entries, starting at “a.” When it reaches a predefined number, it sends the list to the next available node and begins building a new list. If a node returns an entry that contains the answer, then the program will print that the password was cracked and how long it took to crack that entry.

Dictionary attacks are the easiest to set up. This attack uses a pregenerated wordlist – a list of common words, passwords, or combination of letters. The program loads this wordlist and will attempt each entry in the list against the loaded hash. Common wordlists include the top 1,000 children’s names, every word on Wikipedia, and every date from 1900 to the present.

A hybrid attack is designed to catch the passwords that the dictionaries missed, such as “hello312.” It checks every entry in the loaded wordlists, but appending or prepending (or both) the numbers 0-999 to the entry before moving on to the next one. This attack takes substantially longer, but is much more thorough than a standard dictionary attack.

RESULTS
As with any project that takes a familiar concept and forces it into an unfamiliar situation, most of the research dealt with taking a known concept (password auditing program) and introducing it to an unknown arena, cluster/distributed computing. The author has generated the primary body of work through research and software modification, resulting in the following sequence of steps. Here, we arrive at software that will run in a clustered environment.

1. There are three pieces of software required to run C_B:
   - Python 2.7.5
   - ParallelPython (PP), authored by Vitalii Vanovschi
   - OpenSSL (v1.0+)

   All three of these applications will run on the following platforms
   - Red Hat Enterprise Linux (RHEL)
   - CentOS
   - Mac OSX
   - Windows XP+

2. C_B is written in Python 2.7.5 and uses ParallelPython to move data between nodes of the cluster and OpenSSL to handle the encryption of NTLM (Microsoft Windows’ NT Lan Manager) passwords.
   - C_B was designed to be run on Red Hat Enterprise Linux (RHEL) and CentOS Linux systems and was coded specifically for them.
   - RHEL and Fedora architecture Linux requires a special flag to be set in the Python source code before it will allow Python and OpenSSL to communicate; this means Python must be compiled from a source for each machine that will be running the program.
   - In addition to OpenSSL, RHEL also requires the packages openssl-devel, python-devel, and gcc.
   - iptables, the Linux firewall, must have port 60000, the default communication port for PP, unblocked.

3. Mac OSX (based on Berkeley Linux) will run C_B cleanly in its base configuration. OSX 10.8 (current as of this writing) includes Python 2.7 and OpenSSL.

4. Windows has a few more issues than Mac OSX owing to the requirement to compile OpenSSL or use an unofficial binary (installable package) to install it. Windows also does not come with Python natively, so that must be downloaded and compiled, requiring gcc.

5. C_B cracks passwords in the Windows NTLM format, which despite being a retired algorithm, is still used in all versions of Windows, including Windows 8.
   - NTLM passwords are simply hashed with the MD4 algorithm (Ewaida, 2010). Windows uses this algorithm if there is no domain controller available – if there is then, Windows uses NTLMv2, which uses a more involved process and is generated on-the-fly (ISECPartners, 2005; Scambray & McClure, 2008).

6. Python includes an implementation of MD5 natively, but doesn’t include MD4 because of its age. Therefore, in order to create hashes to check against the encrypted passwords, Python needs to communicate with OpenSSL,
which does contain the required libraries. MD4 support can be added to Python inside C_B, and is a direction for future research.

Password_Buster was originally coded as a single thread/single core application, which worked, but was extremely slow. The cluster version developed by the author allows for the use of all CPU cores on a system and on multiple systems, which drops the time required for a job by approximately fourfold before adding in the power from the nodes (the other machines on the network). At 20 nodes, the computational time is quartered again. This means that a crack that originally took 10 minutes on the original single thread program takes about 37 seconds on a cluster with 20 nodes.

Limitations

At the current stage of this research and at the time of this writing, there are three noted limitations to the software.

- The dictionary attack crashes the program if it imports a Chinese character from the dictionary.
- The program currently has no ability to dynamically allocate job numbers, which limits its speed at times.
- Some graphics, designed to make the output easier to understand, perform unreliably.

CONCLUSION

In high-security environments such as government or pharmaceuticals, any type of public-facing portal is a cause for worry. Software such as this can be used as a backup or second testing mechanism for mission-critical logins; once a proposed login matches the password requirements in place, it can be tested again in a few hours. The sentiment “…if we can break it, they can break it,” especially holds true here.

Once fully optimized, cluster computing can be an extremely powerful tool for businesses and hobbyists because of its combined power and its ability to “piggyback” on an existing network setup. Several industries already use distributed computing (chemistry being one; Stanford University runs a form of distributed computing called Folding@Home). Information security will benefit from a faster tool to add to their repertoire.

FUTURE RESEARCH AND VERSIONS

The author’s interest in future research includes three areas of study.

Writing a custom plugin implementation of the MD4 algorithm in C is an exciting direction, and according to testing, compiled C is approximately 30 times faster than a pure Python implementation. This would substantially improve the overall speed of the program.

Another research direction worth looking at is the implementation of Rainbow Tables: great quantities of data of precomputed hashes. The value of Rainbow Tables is most apparent in extremely complex passwords or ones that use different character sets, such as “AsD12 !!” (Genbox, 2012). In this particular case, it would take approximately 72 days to break this password, because the total character set needed to break it would include uppercase letters, lowercase letters, numbers, and symbols for a total of 94 possibilities raised to the power of the character-length of the password (Genbox, 2012). In this case, it’s eight characters; 6,095,689,385,410,816 possible combinations. A Rainbow Table took only 37 minutes to uncover this particular password. Although the Rainbow Tables required more than 2TB of hard drive space, the time that they save is well worth the effort required to store them.

C_B is an ongoing research project, and the current research has tested passwords up to four characters in length. Research will continue with increasing password character length.

ACKNOWLEDGMENTS

The author would like to express appreciation to the Department of Information Systems, & Operations Management, the Miller College of Business, and Ball State University for supporting his research in the Cluster Computing Research Lab.

REFERENCES