Bits & Bytes

Arkansas' Premier Computer Club

September 2022

Bella Vista Computer Club - John Ruehle Center

Highlands Crossing Center, 1801 Forest Hills Blvd Suite 208 (lower level), Bella Vista, AR 72715

Website: http://BVComputerClub.org Email: editor@bvcomputerclub.org

MEETINGS

Board Meeting: September 12, 6pm, in John Ruehle Training Center, Highlands Crossing Center.

General Meeting: September 12, 7pm, "Last Pass Password Manager and related material", presenter Woody Ogden. Password Managers are an essential tool for tracking and managing secure passwords for online accounts and devices with passwords. There will be a video on usage of LastPass and a discussion of other aspects of password management.

We will meet in-person in Room 1001 on the lower level of The Highlands Crossing Center, 1801 Forest Hills Blvd, Bella Vista, or you may attend the meeting on-line via Zoom. Zoom access information is published on our website.

Visitors or Guests are welcome.

Because of COVID-19, we recommend observing any current masking and social-distancing guidelines that may be in effect at the time of the meeting. Consider attending by Zoom if you or others in your family are in a high risk category.

HELP CLINICS

Septermber 3, no Help Clinic (Labor Day wkend)
September 21, 9am - noon at John Ruehle center
Members may request Remote Help on our website at https://bvcomputerclub.org at menu path Member Benefits ►Remote Help .

MEMBERSHIP

Single membership is $25; $10 for each additional family member in the same household.

Join on our website at https://bvcomputerclub.org at menu path Get Involved ►Join/Renew, by mailing an application (from the web site) with check, or complete an application and pay in person at any meeting.

CLASSES

(At BVCC Training Center)

Tuesday, Sept 20, 9am-noon, "Using Windows 10", with Joel Ewing.

Wednesday, Sept 28, 4pm-6pm, "Computer Security for Regular People, Part 1", with Justin Sell.

Advance sign up required for each listed class: For reservations: email to edu@bvcomputerclub.org, or sign up at the General Meeting. Classes are **free to Computer Club members.**

Check the monthly calendar and announcements for any last minute schedule changes at https://bvcomputerclub.org .

NEW OR RETURNING BVCC MEMBERS

We are pleased to welcome the following new members or members returning to BVCC after an absence since last month's newsletter:

Jan Reeves	Linda Schmelig	Rich Atha
Gloria Atha	Bob Etien	Ellen Etien
Norma Staley		

ELECTION OF OFFICERS AT AUGUST GENERAL MEETING

The following officers and Board members were elected at the August General Meeting to begin serving terms starting in September:

- President: Joel Ewing
- Vice President: Woody Ogden
- Secretary: Lori Obrenovich
- Treasurer: Dean Larsen
- Board: Loretta Ostenso

Since Dean Larsen was serving in a Board position that expires at the end of August 2023, the Board will appoint a replacement in September to serve out the remaining year of the Board position vacated when Dean becomes Treasurer in September.

BVCC AND POA GIVE-BACK TUESDAYS, OCTOBER 4

BVCC has been scheduled as a participant in the Bella Vista POA Give-Back Tuesdays for the first Tuesday in October, October 4, at the Highlands Pub & Patio, 1 Pamona Dr., open 11am - 8 pm. BVCC will receive 10% of the food sales on that day as a donation. If you are planning to eat out that week, consider the Highland Pub & Patio on the 4th.

Further details when they are available.

ANOTHER EMAIL FRAUD VARIANT

By Joel Ewing
Bits & Bytes, Sept 2022
President (at) BVComputerClub.org

Does anyone have a telephone provider that has the ability to send an email to you when a voice mail message is received? If your telephone provider provides this service, you might occasionally expect to see emails about voice mail messages. The MagicJack service we have for our Training Center phone service can do this, and other phone providers have similar services, including the ability to transcribe voice mail into text messages. In the case of MagicJack, they will send an audio .WAV file.

Several days ago I received a new type of fraud email -- it may not be that new, but this is the first time I've seen one. The email claimed to be from myself (suspicious) with a Subject of "**VoiceNote**

Transcription Message on August 26, 2022", a text body of "**You received a voice-note from WIRELESS CALLER!**" (also suspicious to have no caller-ID for the source), and an attachment of "**Voicenote-Audio Transcription.htm**".

Unlike a WAV file, HTM files can contain malicious code. Looking at the email source, I found that the MIME email body part containing the HTM file was not sent in the more efficient and common UTF-8 character set, but as a base64 encoded part, making it impossible to visually scan the email source for questionable text content (highly unusual for legitimate email content). This should be more than enough to strongly imply that viewing this HTM file in a browser could be dangerous to your health.

But being curious, I went one step further and used a Linux utility to decode the base64 encoded HTM info in the email into text characters to safely examine its content as text. What that revealed was that all of the functional content was hidden in one long executable javascript program that appears to be designed to generate the actual HTML for the web page from a bunch of hex values and a very long character string of what must be encrypted HTML code. This is a technique used by people who are up to no good and trying to hide it -- either trying to bypass malware detection filters or bypass spam detectors. No way would I dare load this HTM into a browser on anything other than an expendable throw-away system containing no actual user data.

A Google search suggests this is most likely a phishing attempt to trick you into supplying credentials to one or more of your Internet accounts, and that would be the easiest exploit to implement. Some similar emails reportedly try to lead the recipient through a series of failures in retrieving his voicemail, eventually asking for login credentials to help resolve an imaginary problem. With the actual HTML hidden the way it is, this attachment could be even be more nefarious, such as attempting to find and exploit security bugs to extract sensitive data or corrupt data on your computer. The only reason that type of attack is less likely is that it requires the scammer to have more talent and expend more time and effort. Unless you are known to be a high-profile target, the odds are that you will only encounter simpler phishing attacks that attempt to trick you into compromising your own sensitive information.

HASHING FUNCTIONS AND HOW THEY WORK

By Joel Ewing
Bits & Bytes, Sept 2022
President (at) BVComputerClub.org

Why Hashing?

One finds references to hash functions or hash values in a number of contexts. Websites with sensitive software downloads will often provide a "hash value" that may be used to verify that that the downloaded software file contains no transmission errors. Installation DVD images for Operating Systems like Linux may include support for media verification that reads the media and checks that its computed hash value is correct to verify that the DVD can be read by your DVD drive without errors.

Blockchain technology, used for cryptocurrency and other distributed ledgers, uses a hash function of the contents of a block to insure that the contents of blocks are not changed after they have been recorded.

Websites that support an account login shouldn't (if they know what they are doing) store actual passwords on the website, but instead store a hashed version of the password. That way if there is a mass theft of website data, the actual passwords of users aren't revealed, and in fact the maintainers of the website don't have access to the actual passwords either. A user-entered password is similarly hashed and compared with the retained hashed value to see if a login should be allowed.

Hashing and encryption are related in that both techniques disguise the original data, and some hashing techniques even use algorithms related to encryption algorithms. The difference is that encryption by design is intended to be reversible, provided you have access to the decryption process and decryption keys. "Hashing" is in general an irreversible process, even if the hashing technique is known. Recovery of an original value from a hash value in general requires an exhaustive search (which is another reason why one should still never use a password from the "worst" passwords list for anything)

Hashing can also be used as a way to spread out entries in an in-memory table in a way that allows the entries to be quickly found when needed. When used in this fashion, the object is not to generate a unique ID, but to map a data value used for a search into a numeric value in the range [0, n-1] for use an in index into a table with space for "n" entries.

Creating An Almost-Unique ID To Verify Data Has Not Been Changed

This usage of a hashing function reduces a string of data of variable length to a single numeric value of fixed length

f(*data*) -> *hash_value*

There are multiple uses for this concept, but one of the most common is to compute an abbreviated representation of the data that will be practically unique and that can be used to confirm that the data has not been changed. Examples of hashing functions for that purpose are SHA1 (generates a 160 bit hash value) and SHA256 (generates a 256 bit hash value). If re-computing the hash value for the data produces the same hash value as previously, it is almost impossible that the data has been changed.

Clearly if the data is longer than (has more bits than) the hash value, then mathematically there must be cases where different data strings map to the same hash value; but in practice it would be very difficult to exploit that. There are very complex rules for valid bit or character sequences in the data: language syntax and semantics for text, restricted formats for numeric values, some fields limited to a small set of values, etc. All those factors make it very unlikely that two different valid data values would map to the same hash value. Hash functions used for data validation are designed so that it is practically impossible to modify the data accidentally or deliberately in useful ways that would leave the hash value intact. If meaningful changes were made to one part of the data, it might in theory be possible to find some way to make offsetting changes somewhere else in the data that would preserve the hash value, but it's very unlikely such offsetting changes would also result in valid data values.

With SHA256 the hash value has 2^{256} possible values, which is over 1.15 x $10^{77.}$, You would have to have a ridiculously impossible number of different data values[1] before there is a certainty that a well-designed hash

1 The number of distinct 256-bit hash values is massively more than the total number of atoms on Earth, which is estimated at only 1.33 x 10^{50}, so we couldn't come even remotely close to constructing any devices to store that many distinct data values using any technology we can currently imagine.

function would map two different data values to the same hash value. That doesn't mean a collision with fewer data values is impossible , but it is highly improbable -- and even less likely that it could be usefully exploited.

You will sometimes see such hashing functions referred to as a crypto-hash, suggesting a relationship with cryptography; but while such numeric IDs are certainly cryptic, there is no intent that these values have a practical use to secretly communicate previously unknown data to another party. There is simply no way to reverse the calculation to produce the original data string without much more information than the hash value. If the recipient had an agreed set of messages to expect and knew a hash value corresponded to one of those messages, they could then pick from that list the message that mapped to the sent hash value, but that would be essentially the same as a substitution cipher where both parties agree in advance that each possible message would be represented by some unique sequence of symbols, with a code mapping table in the hands of both parties.

Hashing As a Fast Way To Look Up a Value in a Table

A very different usage of hash functions is for efficient location of a table entry corresponding to a key value. Looking up an entry in a table is a common process required in many different computer applications. When a table is very large, or even for smaller tables that are searched a very large number of times, choosing the most efficient way to search the table can make the difference between success or failure of the application.

Think of a table search as something like a dictionary lookup.   You have a relatively short string, the "key" or "word" to find, a table containing entries for all the known key values, and you need to access the information associated with a given key.   This is an extremely common task used in many applications, both manual and computerized. It could be an actual dictionary, a list of all account login names with all the related account information as data, a list of all State names with their corresponding two-character postal abbreviation, a list of all postal zip codes mapped to a city name, etc.

The simplest, brute-force approach for finding the desired entry in a table is to start at the first table entry. If that entry has the desired key, the desired entry has been found; if not advance to the next entry and repeat, until the desired entry is found.   The efficiency of this sequential-search approach gets proportionally worse as the number of entries increases.   You would never think of manually searching an English dictionary with tens of thousands of words using this method, and a heavily used search in a computer application should never be using this technique on an in-memory table with over 50 entries.

When manually searching a large book like a dictionary, where word entries are in alphabetical order, you instead make an educated guess about what page to check first based on the first letter(s) in the word being sought. Depending on the contents of that page, the desired entry is either on the current page, or by the alphabetic ordering you have eliminated either all other pages before the current page or all pages after the current page.   If the correct page has not been found, you then take the part of the book that has not been eliminated and repeat the process with another educated guess.   One computer equivalent of this manual process is called a "binary search", where the "educated guess" of what entry to check next is the entry in the middle of the portion of the table that hasn't yet been eliminated. This is much more efficient than a sequential search for large tables: binary search of a table of 511 entries requires checking only 8 entries in the worst case, and an average across all entries of slightly under 8; a sequential search of 511 entries in the worst case requires checking all 511 entries, and the average across all entries is to check 256 entries -- 64 times more resource-expensive than a binary search.   The larger the table, the worse the comparison becomes.

For extremely large tables, or tables that are searched millions of times, hash function techniques can be used for even faster access of a desired entry. Some programming languages even provide direct support for table storage by-key, using hashing techniques to find entries without the programmer having to be aware that hashing is used under the covers.

In the best case, it would be nice to have a simple hashing function that directly transforms each "key" into a unique index that takes you directly to the desired table entry:

  f(*key*) -> *table_index*

so that only a single table entry must be examined, the entry that is needed. It is only in some very specialized cases where all possible key values are known in advance that the uniqueness of the mapping for all keys can be assured, but it is usually possible to find some relatively simple hashing function that performs well enough to keep the number of non-unique hash values acceptably small.

Typically, a table that will be accessed using hashing techniques will be deliberately over-sized, so that when all entries have been stored the table will still only be 50% to 90% full. As the percentage of the table entries in use increases, the odds increase that two different keys will be found that map to the same table index, a condition called a "collision". The table loading and search process must be designed to handle such collisions by verifying that an accessed entry is either empty or has the correct key, and if not, then having consistent rules for where to search next until the correct entry is found. Various techniques can be employed so that as long as the table-in-use percentage doesn't get too high, the average number of entries that must be checked for each search is somewhere between 1 and 2, ***independent of the total entries in the table***. All other table searching methods get more expensive as the number of entries in the table grows.

Some very simplistic hash functions can be successfully used for this purpose. The key values are first reduced to a integer numeric value, which when divided by the physical table size "n" leaves a remainder (0 to n-1) that can be used as the index of the first table entry to be checked. To make it more likely this will spread key entries uniformly across the table, the value for "n" should be chosen as a prime number just sufficiently large enough to keep the "full" table residency in the desired percentage range. If too many hash collisions reduces the efficiency of the search process, one way to improve the efficiency is just to increase the size of the table to reduce the occupancy percentage and reduce collisions. In other words, using more memory can reduce the search time.