

Tutorial: John the Ripper - Why You Are Doing It Wrong

By Thomas Wilhelm, ISSMP, CISSP, SCSECA, SCNA

Many people are familiar with John the Ripper (JTR), a tool used to conduct brute force attacks against local passwords. The application itself is not difficult to understand or run... it is as simple as pointing JTR to a file containing encrypted hashes and leave it alone. In a professional penetration test, we don't always have the time to allow JTR to run to completion, and we must rely on some additional techniques to speed things up including the use of wordlists or dictionaries. JTR comes with its own wordlist containing supposedly common passwords, and we can use that dictionary to identify some low-hanging fruit. However, in most cases, the supplied JTR wordlist is woefully inadequate in identifying a wide-range of commonly-used passwords, especially when people prefer to select passwords that have some meaning to them (e.g. hobbies, partner names, child names, and pet names). So how can we improve our use of JTR to catch passwords that have relevancy to the users of our target system? It may be a bit more complicated than it seems.

The Information Systems Security Assessment Framework (ISSAF) provides an adequate methodology when focusing on password attacks and includes the suggestion of using dictionaries. For those who conduct penetration testing, the use of dictionaries is only one of two prongs used in attacking a local, encrypted password list; brute force attacks are conducted after we have attempted to break passwords using dictionaries. In this fashion, we can (hopefully) obtain weak passwords to work against during the pentest; anything discovered during the brute force attack (assuming it is too late in our pentest to use then) can simply be added to our wordlist for future penetration test projects.

del.icio.us

Discuss in Forums {mos_smf_discuss:/root}

Train at the Hacking Dojo with Thomas Wilhelm - Advertisement

To summarize the ISSAF, the following needs to be added to any working wordlist (OISSG, 2006, p. 230):

- Small international (English) and medium local (ex. Italian) dictionaries
- Information gathered
- Formatted and unformatted dates starting from 60 years ago
- The name of soccer/football/basketball teams, the name of notorious TV people
- Users register codes
- Etc.

Although additional information could certainly be included in this list (vernacular used in the users' hobbies comes to mind immediately), it is a good starting point and makes sense. However, there are some hidden problems not often discussed in using the wordlists suggested by the ISSAF - the problem is encoding.

Let's take a look at an example using John the Ripper against a common word. In Figure 1, we see two different SHA-1 hashes that were computed for a single word (which should be theoretically impossible, but we will get to that soon). When we run JTR against the two hashes, we see that JTR was able to properly identify one of them correctly, but not the other. If the assertion that both hashes are for the same word is true, and yet they are distinctly different, something must have happened during the encryption process that changed our word before presenting us with the encrypted value.

Figure 1 - SHA-1 Hashes

At this point, we can see that the word we cracked was German in nature; more importantly, it contained non-ASCII characters, specifically a "Latin Small Letter U with diaeresis." This translates to Unicode value "U+00FC" and seems to have been retained in the user2 encryption process, since JTR was able to crack the hash. The mystery seems to be as to the changes that occurred with the user1 password. To make sense of the discrepancy between the two hashes that had used the same word for their input, let's examine what Unicode is, and its purpose.

The Unicode Consortium has developed a universal character set, which "covers all the characters for all the writing systems of the world" (Unicode.org). With regards to programming, the UTF-32 protocol requires four bytes for each character, which makes it easier to manage storage; however, other versions of UTF use different byte sizes making storage and transmission of Unicode somewhat problematic (or at least requires some forethought). Because of byte size and the fact Unicode is not byte oriented (excluding UTF-8), programmers have sometimes opted to convert Unicode into something easier to manage; it seems the most common encoding schema used to convert Unicode over the years is base64, which consists of the character set a-z, A-Z, and 0-9 (plus two additional characters) (Network Working Group, 2003). Base64 is used in numerous applications already, and many different routines exist to convert Unicode into base64.

So, what happens to our Unicode German word if converted into base64 and then back into plaintext? The word is transformed into "Glückwunsch" - the "ü" has been replaced with "ü." Once we understand that the word has been seriously mangled, we realize that the only way for John the Ripper to convert this value for us would be through brute force, and, considering the string length (16 characters), we may never have enough time to dedicate to its eventual discovery. What is worse, is that "Glückwunsch" is a fairly common word in German, which could easily be defined as "low hanging fruit"

assuming that we used a medium-sized German dictionary to use as part of our initial crack attempt. To avoid missing such an easy word, we have two alternatives - identify those applications that convert Unicode into base64, or expand our wordlist to include base64-translated characters.

So, how do we identify applications that convert Unicode into base64? Unfortunately, there are no reliable methods to do so. The only clue we can rely on is if base64 has had to add fillers, which can be distinguished by the equal sign ("="). As an example, the word "Glückwunsch" encoded into base64 is "R2wmlzl1Mjtja3d1bnNjaA==" (without quotes). The equal signs are used to pad the actual base64 value until the string is a multiple of 4. However, this assumes we can see the base64 value, before it is placed through the encryption algorithm. In the example seen in Figure 1, there is no way to tell if either of the hashes had base64 or Unicode characters processed through the SHA-1 algorithm. This leaves us with the unfortunate duty of transposing Unicode characters into base64 equivalents in our wordlists.

In Figure 2, we can see a wordlist only containing the German word "Glückwunsch" with both the Unicode version and the base64->text version. Once we run John the Ripper against our original SHA-1 hashes using the new dictionary, we see that we were able to successfully crack both hashes.

Figure 2 - German Dictionary with Unicode and ISO 8859-1 Characters

What does this mean in a real-world penetration test? If our target system has users on it that use a language with special characters, then we may be missing passwords that should be easily cracked, unless we modify our "local" (language-specific to the users) wordlist. The good news is that we only have to make the additions once, assuming we retain our dictionaries over time. The bad news is... you have to break out your scripting skills to make this task easier (I can hear the "boos" already).

And a bit of more bad news... because of my habit to include within all my articles some method of improving one's skillset (which my students affectionately call "homework"), I have a very simple challenge for you, the gentle reader (which must be on my desk no later than next Monday morning). The following are hashes of different French words; however, I will not tell you if they were encrypted with Unicode or base64 values. Feel free to try them out and see how well you do:

\$1\$flh2D0vJ\$Yv4LbmtbSCyOFCK/ffOF51

cUqkZh8CmoMPk

\$H\$9fSsh3tQq4oyXBw/BgQq.q5Q36SdZq0

2c673242281f3419

dG91am91cnM=

Again, these are extremely simple, so you should be able to crack all of them. If you have any problems, feel free to leave a comment here at EthicalHacker.net, or you can contact me directly at: [twilhelm \[at\] heorot \[dot\] net](mailto:twilhelm@heorot.net). Thanks, and

enjoy!

Thomas Wilhelm has been involved in Information Security since 1990, where he served in the Army for eight years as a Signals Intelligence Analyst / Russian Linguist / Cryptanalyst.

A speaker at security conferences across the U.S., including DefCon, HOPE, and CSI, he has been employed by Fortune 100 companies to conduct Risk Assessments, participate and lead external and internal Penetration Testing efforts, and manage Information Systems Security projects.

Thomas is also a Doctoral student who holds Masters degrees in both Computer Science and Management. Additionally, he also dedicates some of his time as an Associate Professor at Colorado Technical University, and has contributed to multiple publications, including both magazines and books. His latest contribution was the publication titled "Professional Penetration Testing," released in August, 2009, which was his fourth book contribution to Syngress. You can also find him training both military and civilian personnel at <http://www.heorot.net> and <http://www.hackingdojo.com>.