Security exercise: hashbreaking

Troels Sørensen, Maurice Mugisha, Rasmus Pommer,
Niels Christensen & Niels Abildgaard

March 13, 2017

1 Introduction

The following is a report of our actions and results while trying to break some 1022 sha224 hashes.

In this quest we were competing with other students of the Security course at the IT University of Copenhagen in spring 2017. The winners, all students with an equal number of broken hashes, would get a Ritter Sport chocolate of their choosing. This reward was simply too high to go at half-heartedly.

We decided to band together in a group, both for generating a broader spectrum of ideas, and also to be able to try more approaches at once.

2 Breaking easy hashes

To gauge the general difficulty we started out with using an online service, CrackStation, to see how many hashes we could break. CrackStation is basically a rainbow table as a service, and through distributing manual labor between all members of the group, we quickly cracked the first 1000 or so hashes.

Rainbow tables are great! They save us a lot of energy when cracking passwords that are at least somewhat commonly used, and have been leaked in unsalted form. The main limiter to the manual approach was filling out reCAPTCHAs, of which we had to fill out less than 55 in total (CrackStation has a limit of looking up at most 20 hashes at a time).

The last few hashes, however, were not found on CrackStation... we needed a different approach.

3 Phishing for solutions to hard problems

We were pretty much out of ideas for how to get hold of the last hashes. At the same time, we were not feeling very confident that what we had cracked so

```
<form action="/i-want-passwords.php" method="post">
    <label for="password">
        password
        <input type="password" name="password">
    </label>
    <button type="submit">get passwords.dec</button>
</form>
```

Figure 1: The code setting up a simple HTML form in *index.html* on the phishing site.

```
if (isset($_POST['password'])) {
  $log = './log';
  $pw = htmlspecialchars($_POST['password'],
          ENT_QUOTES, 'utf-8');
  file_put_contents($log, $pw.PHP_EOL ,
          FILE_APPEND | LOCK_EX);
}
```

Figure 2: The backend code logging password submissions to a log file in *i-want-passwords.php*.

far would yield any chocolate. We decided on implementing a phishing attack aimed at teachers and fellow students.

The hope was that we might get a single code or so from a TA or from the professor, and that students who had cracked more codes than us might fall for our attempt as well.

The phishing attack took the form of a simple form (figure 1), posting to a PHP page, which would save the posted data and respond "Wrong password. Try again." (figure 2).

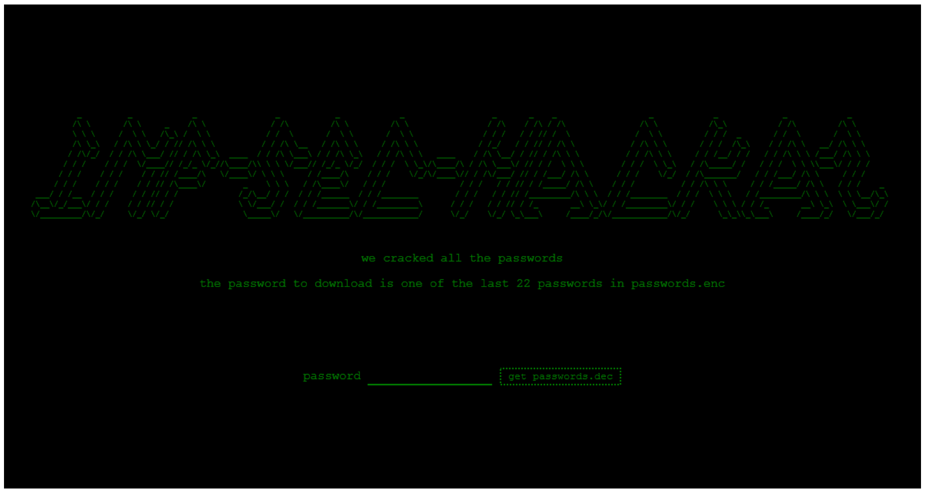
The design of the page was sufficiently *hackery* to look legitimate for the purpose: how would elite-hacker students who had solved the entire problem and wanted to disrupt the system by sharing it present themselves? Pretty much like what you see in figure 3.

We shared a link to the site (hosted on Troels' private server) on the internal class forums under a sufficiently 1337 pseudonym: "p4ZZw0rd m0nst4".

Shortly after posting, Søren, the instructor on the course, responded:

> I think there's some error in your webpage, because I tried the password for item 1022, which is 13afe802c702da826970aa41, but the webpage just says "wrong password". ??!?!?? Please help! Love, Søren

We read this as an obvious act pretending to fall for the phishing attempt.


we cracked all the passwords

the password to download is one of the last 22 passwords in passwords.enc

Figure 3: The visual, *hackery*, appearance of our phshing site.

While it did give us a valid key, Søren also exposed it to the rest of the students giving us no advantage.

Troels was identified by name as the owner of the server. We are still not entirely clear on how this was done, but there are several possible methods.

Shortly thereafter the server went down.

We first assumed that this was the work of the TAs, but later learned that another team of students had decided to perform a DDoS attack on the server (see section 3.1). From their perspective this makes sense: in order to win chocolate, they would need to keep us from having cracked more hashes than they had.

Unfortunately, this stopped us from the second phase, we had planned for the phishing attack: using social engineering and sowing doubt. By first having one member of the group post to LeanIT (which, unlike the Ublend forum is not anonymous) mentioning the site, and then having another group member respond along the lines of "wow, it works!" we hoped to get more people to risk it and give us some passwords. We never know whether this would have worked: a single vigilant competing group prevented this angle.

To avoid tying the server to any person of the group we considered using some free online hosting service, signing up under a false name and using a throw-away email, such as those provided by `lortemail.dk`. It is likely that such an approach would also have limited the damage done by an attempted DDoS attack, as a larger server provider would have greater resilience than the virtual private server we used. To be honest, though, we had not expected at DDoS attack, and so didn't do anything to protect against it at all.

3.1 Analysis of the DDoS-attack

The attack mainly came from two different IP's, 212.71.238.108 and 5.157.7.114. According to ip-lookup.net they are located in the Netherlands and in Singapore respectively. This makes it unlikely that it was the TAs, but we never know.

The attack seems to be automated, since the paths being tried were random two letter file names (it.php, ir.php, ki.php) and file names related to login (access.php, account.php). Some of the attack had the Browser Agent set to "Jakarta Commons-HttpClient/3.1" which is a java library from Apache that can be used to automate such an attack.

The result of the attack was that the server stopped responding, effectively a DDoS attack. The attack was stopped by setting a rule in the firewall preventing requests from those IP addresses.

4 Hopeful brute-forcing

As a last-ditch effort we set up a naïve program to attempt brute-forcing the last hashes, assuming some of them would be hashes of hex values (the phished plaintext was), but we were ultimately unsuccesful.

5 Conclusion

Many of the hashes were available in rainbow tables, a single one was phished for, but our phishing attack was shut down before we got to try all angles on it. We also tried brute-forcing the hard hashes, but were once again unsuccesful.

In the end, many members of the group forgot to hand in the hashes that we did break, limiting the possible number of chocolates we might get. Oh well. We had fun.