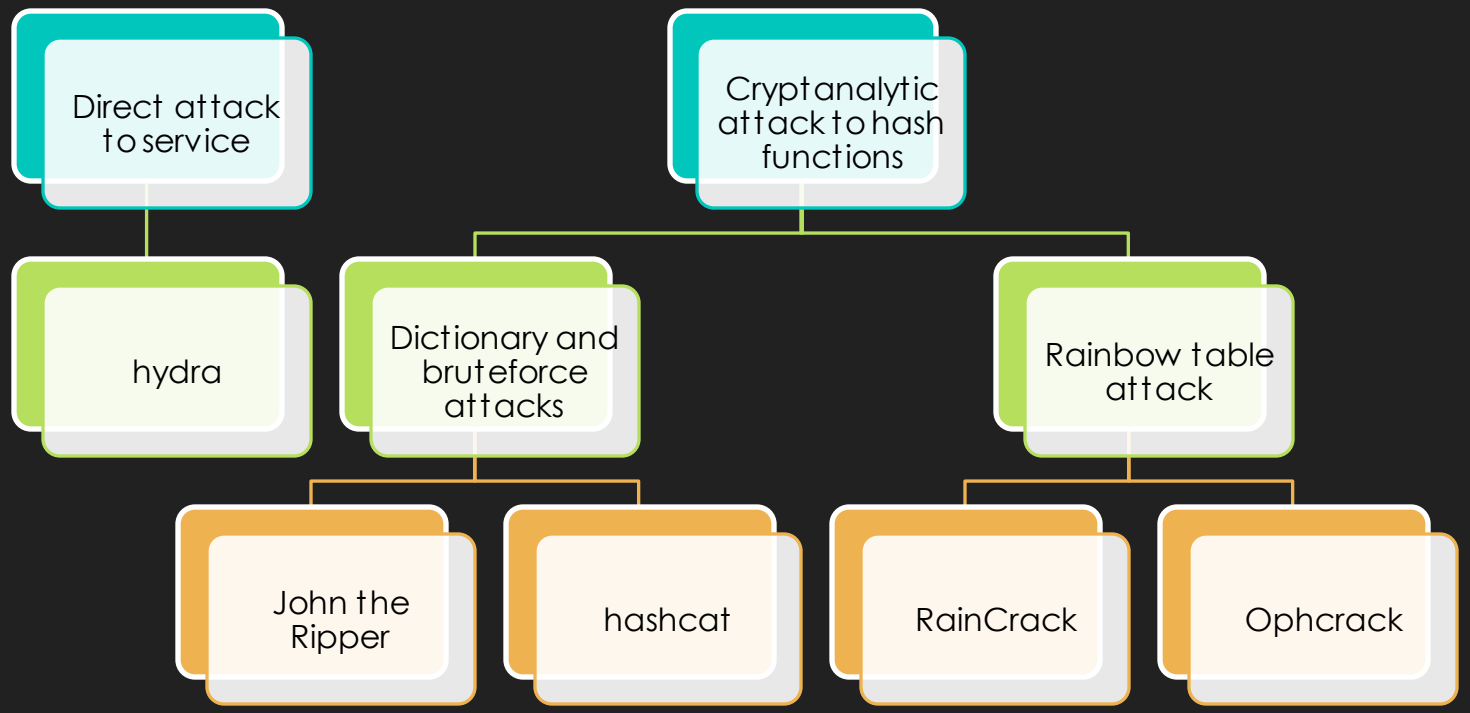
Pratical intruduction to hash functions and password cracking

Direct attacks, properties of hash functions, guessing attacks, rainbow tables

Matteo Cavallaro

Types of attack and tools



Crunch

- Crunch is a wordlist generator where you can specify a standard character set or a character set you specify. crunch can generate all possible combinations and permutations.
- This code can be easily adapted for use in brute-force attacks against network services or cryptography.

```
crunch 3 7 abcdef
```

Crunch – some options

- `-d [n][@,%^]:` limits the number of duplicate characters
- `-p charset:` tells crunch to generate words that don't have repeating characters
- `-t @,%^:` specifies a pattern, eg: `@@god@@@@`

- @ will insert lower case characters
- , will insert upper case characters
- % will insert numbers
- ^ will insert symbols

Hydra

- a parallelized login cracker
- supports numerous protocols to attack
- very fast and flexible, new modules are easy to add
- supports: Asterisk, Cisco AAA, Cisco auth, Cisco enable, CVS, FTP(S), HTTP(S)-FORM-GET, HTTP(S)-FORM-POST, HTTP(S)-GET, HTTP(S)-HEAD, HTTP-Proxy, ICQ, IMAP, IRC, LDAP, MS-SQL, MySQL, NNTP, Oracle Listener, Oracle SID, PC-Anywhere, PC-NFS, POP3, PostgreSQL, RDP, Rexec, Rlogin, Rsh, SIP, SMB(NT), SMTP, SMTP Enum, SNMP v1+v2+v3, SOCKS5, SSH (v1 and v2), SSHKEY, Subversion, Teamspeak (TS2), Telnet, VMware-Auth, VNC and XMPP.

```
hydra [[[-l LOGIN|-L FILE] [-p PASS|-P FILE]] | [-C FILE]] [-t TASKS] [-x
MIN:MAX:CHARSET] [service://server[:PORT]]
```

Cryptographic hash function

- It is a mathematical algorithm that maps data of arbitrary size (often called the "message") to a bit string of a fixed size (the "hash value", "hash", or "message digest") and is a one-way function, that is, a function which is practically infeasible to invert.
- Ideally, the only way to find a message that produces a given hash is to attempt a brute-force search of possible inputs to see if they produce a match, or use a rainbow table of matched hashes.

Cryptographic hash function

- it is deterministic, meaning that the same message always results in the same hash
- it is quick to compute the hash value for any given message
- a small change to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value (avalanche effect)

Resistance properties

Pre-image resistance

- Given a hash value h it should be *difficult* to find any message m such that $h = \text{hash}(m)$. This concept is related to that of a one-way function.

Second pre-image resistance or *weak collision resistance*

- Given an input m_1, it should be *difficult* to find a different input m_2 such that $\text{hash}(m_1) = \text{hash}(m_2)$.

Collision resistance or strong collision resistance

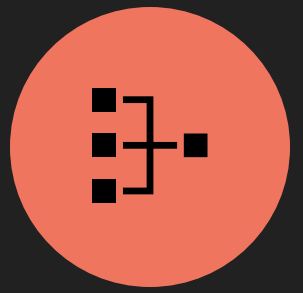
- It should be *difficult* to find two different messages m_1 and m_2 such that $\text{hash}(m_1) = \text{hash}(m_2)$.

Applications

Verifying the integrity of messages and files

Signature generation and verification

Password verification



Proof-of-work system

Some algorithms

- MD5 was designed by Ronald Rivest in 1991 (RFC 1321). Collisions against MD5 can be calculated within seconds which makes the algorithm unsuitable for most use cases where a cryptographic hash is required.
- SHA-1 was developed as part of the U.S. Government's Capstone project. The original specification – now commonly called SHA-0 – of the algorithm was published in 1993 by NIST. It was withdrawn by the NSA shortly after publication and was superseded by the revised version, commonly designated SHA-1.
- Bcrypt (1999)
- Whirlpool (2000)
- SHA-2 is a set of cryptographic hash functions designed by the NSA, first published in 2001.
- SHA-3, released by NIST on August 5, 2015
- BLAKE2 and BLAKE3

John the Ripper

- fast password cracker
- available for many flavors of Unix, macOS, Windows, DOS, BeOS
- its primary purpose is to detect weak Unix passwords
- supports also Kerberos/AFS, Windows LM and NTLM, various macOS and Mac OS X user password hashes, MD5, SHA-1, SHA-256, and SHA-512 and many others
- built-in parallel processing support using OpenMP

Wordlist mode

- Dictionary attack
- Optional word mangling rules (which are used to modify or "mangle" words producing other likely passwords)

```
john --wordlist=password.lst [--rules] mypasswd
```

"Single crack" mode

- It will use the login names, "GECOS" / "Full Name" fields, and users' home directory names as candidate passwords, also with a large set of mangling rules applied.
- much faster than wordlist mode

```
john --single mypasswd
```

"Incremental" mode

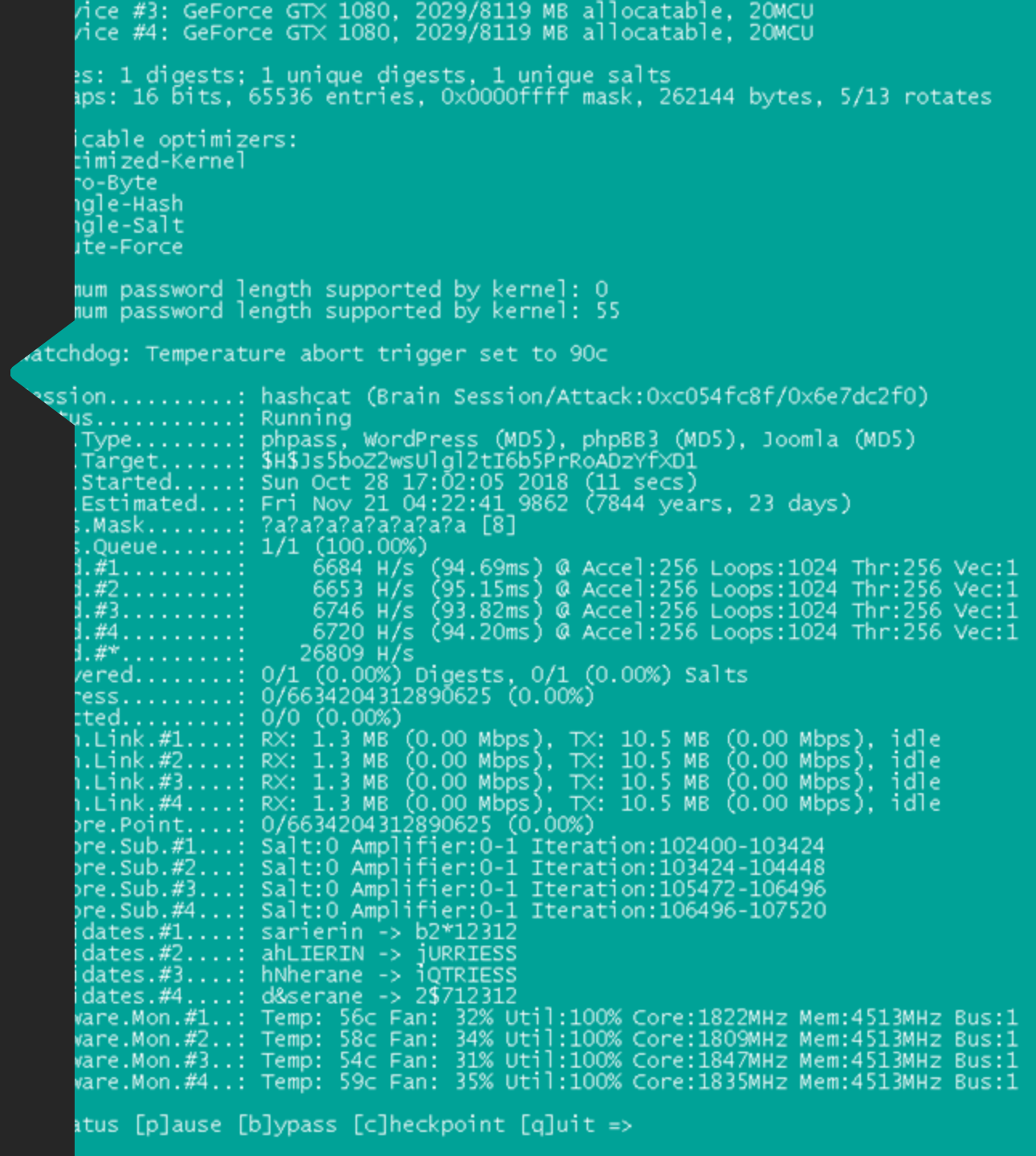
- the most powerful cracking mode
- it can try all possible character combinations as passwords
- you need a specific definition for the mode's parameters, including password length limits and the charset to use

```
john –incremental=charset mypasswd
```

- pre-defined incremental modes:
  - "ASCII" (all 95 printable ASCII characters)
  - "LM_ASCII" (for use on LM hashes)
  - "Alnum" (all 62 alphanumeric characters)
  - "Alpha" (all 52 letters)
  - "LowerNum" (lowercase letters plus digits, for 36 total)
  - "UpperNum" (uppercase letters plus digits, for 36 total)
  - "LowerSpace" (lowercase letters plus space, for 27 total)
  - "Lower" (lowercase letters), "Upper" (uppercase letters)
  - "Digits" (digits only).

hashcat

- World's fastest password cracker
- Open-Source (MIT License)
- Multi-OS (Linux, Windows and macOS)
- Multi-Platform (everything that comes with an OpenCL runtime)
- 200+ Hash-types implemented



Supported attack modes

- Dictionary attack
- Combinator attack
- Brute-Force attack (subcase of Mask attack)
- Mask attack
- Hybrid attack
- Rule-based attack
- Toggle-Case attack (subcase of Rule-based attack)

Dictionary Attack

- The dictionary attack, or “straight mode,” is a very simple attack mode. It is also known as a “Wordlist attack”.
- All that is needed is to read line by line from a textfile (aka “dictionary” or “wordlist”) and try each line as a password candidate.

```
cat dict.txt | hashcat -a 0 hash.txt
```

Combinator Attack

- Each word of a dictionary is appended to each word in a dictionary.

```
hashcat -a 1 hash.txt dict1.txt dict2.txt
```

Mask Attack

- Try all combinations from a given keyspace just like in Brute-Force attack, but more specific.

```
hashcat -a 3 hash.txt mask
```

Mask Attack – charsets

Built-in charsets

- `?l = abcdefghijklmnopqrstuvwxyz`
- `?u = ABCDEFGHIJKLMNOPQRSTUVWXYZ`
- `?d = 0123456789`
- `?h = 0123456789abcdef`
- `?H = 0123456789ABCDEF`
- `?s = «space»!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~`
- `?a = ?l?u?d?s`
- `?b = 0x00 - 0xff`

Custom charsets

- `--custom-charset1=CS`
- `--custom-charset2=CS`
- `--custom-charset3=CS`
- `--custom-charset4=CS`

These command line-parameters have four analogue shortcuts called

`-1`, `-2`, `-3` and `-4`.

Mask Attack – example

command	keyspace
-a 3 ?l?l?l?l	aaaa – zzzz
-a 3 -4 ?l?d ?4?4?4?4	aaaa – 9999
-a 3 password?d	password0 – password9
-a 3 -4 ?l?u ?4?l?l20?d?d	aaa2000 – Zzz2099

Hybrid Attack

- Basically, the hybrid attack is just a Combinator attack. One side is simply a dictionary, the other is the result of a Brute-Force attack. In other words, the full Brute-Force keyspace is either appended or prepended to each of the words from the dictionary. That's why it's called "hybrid".

```
hashcat -a 6 example.dict ?d?d?d?d
hashcat -a 7 ?d?d?d?d example.dict
```

```
password0000        0000password
password0001        0001password
password0002        0002password
.                   .
.                   .
password9999        9999password
hello0000           0000hello
hello0001           0001hello
hello0002           0002hello
.                   .
.                   .
hello9999           9999hello
```

Rule-based Attack

one of the most complicated of all the attack modes

like a **programming language** designed for password candidate generation

has functions to modify, cut or extend words and has conditional operators to skip some, etc.

the most flexible, accurate and efficient attack.

Why re-invent the wheel? Regular expressions are too slow.

Rainbow table

- A **rainbow table** is a precomputed table for reversing cryptographic hash functions.
- It is a practical example of a space–time tradeoff, using less computer processing time and more storage than a brute-force attack which calculates a hash on every attempt, but more processing time and less storage than a simple lookup table with one entry per hash.
- introduced by Philippe Oechslin in a 2003 research paper
- ineffective against one-way hashes that include large salts
- saltedhash(password) = hash(password | | salt)

Making a Faster Cryptanalytic Time-Memory Trade-Off

Philippe Oechslin

Laboratoire de Securité et de Cryptographie (LASEC)
École Polytechnique Fédérale de Lausanne
Faculté I&C, 1015 Lausanne, Switzerland
philippe.oechslin@epfl.ch

Abstract. In 1980 Martin Hellman described a cryptanalytic time-memory trade-off which reduces the time of cryptanalysis by using precalculated data stored in memory. This technique was improved by Rivest before 1982 with the introduction of distinguished points which drastically reduces the number of memory lookups during cryptanalysis. This improved technique has been studied extensively but no new optimisations have been published ever since. We propose a new way of precalculating the data which reduces by two the number of calculations needed during cryptanalysis. Moreover, since the method does not make use of distinguished points, it reduces the overhead due to the variable chain length, which again significantly reduces the number of calculations. As an example we have implemented an attack on MS-Windows password hashes. Using 1.4GB of data (two CD-ROMs) we can crack 99.9% of all alphanumerical passwords hashes (2^{37}) in 13.6 seconds whereas it takes 101 seconds with the current approach using distinguished points. We show that the gain could be even much higher depending on the parameters used.

Key words: time-memory trade-off, cryptanalysis, precomputation, fixed plaintext

1 Introduction

Cryptanalytic attacks based on exhaustive search need a lot of computing power or a lot of time to complete. When the same attack has to be carried out multiple times, it may be possible to execute the exhaustive search in advance and store all results in memory. Once this precomputation is done, the attack can be carried out almost instantly. Alas, this method is not practicable because of the large amount of memory needed. In [4] Hellman introduced a method to trade memory against attack time. For a cryptosystem having N keys, this method can recover a key in $N^{2/3}$ operations using $N^{2/3}$ words of memory. The typical application of this method is the recovery of a key when the plaintext and the ciphertext are known. One domain where this applies is in poorly designed data encryption system where an attacker can guess the first few bytes of data (e.g.

How chains work

$$\texttt{aaaaaa} \xrightarrow[H]{} \texttt{281DAF40} \xrightarrow[R]{} \texttt{sgfnyd} \xrightarrow[H]{} \texttt{920ECF10} \xrightarrow[R]{} \texttt{kiebgt}$$

$$\texttt{920ECF10} \xrightarrow[R]{} \texttt{kiebgt}$$

$$\texttt{aaaaaa} \xrightarrow[H]{} \texttt{281DAF40} \xrightarrow[R]{} \texttt{sgfnyd} \xrightarrow[H]{} \texttt{920ECF10}$$

$$\texttt{FB107E70} \xrightarrow[R]{} \texttt{bvtdll} \xrightarrow[H]{} \texttt{0EE80890} \xrightarrow[R]{} \texttt{kiebgt}$$

RainbowCrack

is a general propose implementation of Philippe Oechslin's faster time-memory trade-off technique

runs on Windows (with GPU acceleration support) and Linux

supports LM hash, NTLM hash, MD5, SHA-1, SHA-256

extensible with plugins

Table generation

```
rtgen  hash_algorithm  charset  plaintext_len_min  plaintext_len_max
table_index  chain_len  chain_num  part_index
```

- `table_index:` selects the reduction function
- `chain_len:` the rainbow chain length. Longer rainbow chain stores more plaintexts and requires longer time to generate.
- `chain_num:` number of rainbow chains to generate

Last steps: `rtsort`, `rtmerge`, `rt2rtc`

rcrack

Basic usage

- `rcrack[_cuda | _cl] <table directory> -h <hash> | -l <hash list file>`

There are also

- `rcrack_gui, rcrack_cuda_gui, rcrack_cl_gui`

Ophcrack

- Free and open source (GPL license)
- Multi-OS (Windows, Linux/Unix, Mac OS X)
- Cracks LM and NTLM hashes
- Real-time graphs to analyze the passwords
- Developed by Philippe Oechslin (and others)

Example table for LM hashes

- alfanumeric charset
- 99.95% success rate
- size of 388MB
- average number of hash operations: 15.3 million
- maximum number of hash operations: 297 million