Jméno a příjmení: **Adam Šárek**
Osobní číslo: **SAR0083**
Datum: **4.11.2022**

Forenzní analýza - Protokol (6)

Problematika prolamování hesel

1. **Seznamte se s problematikou hesel**
2. **Seznamte se s problematikou prolamování hesel**
3. **Seznamte se s problematikou složitosti hesel a heslové politiky**

4. **Na základě získaných a nastudovaných informací proveďte tyto úkoly:**
    a. Udělejte jednoduchou (webovou aplikaci), která bude obsahovat formulář s heslem a volbou složitosti hesla, které prolamujete
        i. Formulář bude odesílat data metodou POST
        ii. Složitosti(pro srovnání použijte vzorové hesla v závorce):
            0 = 4 čísla [5612]
            1 = 5 čísel [32276]
            2 = 4 písmena [bzcd]
            3 = 4 písmena a čísla [h4b2]
            4 = 4 písmena a čísla a speciální znaky [z+1c]
        iii. Pokud heslo je správne stránka po odeslání formuláře ukáže “OK”, pokud špatné tak “KO”
    b. Heslo se pokuste prolomit pomoci javascriptu (AJAX) a dalšího programu v jazyce (Python, C, C++, C#, JAVA, atd). Pro generování všech možných kombinací můžete použít knihovnu.
    c. Oba přístupy k prolamování srovnejte a odůvodněte. Uveďte čas, který byl potřebný k prolomení formuláře pro výše uvedené vzorové hesla. (Uveďte i hardware)
    d. Implementujte do svého formuláře / stránky ochranu proti BruteForce
    e. Pokuste se vašim a nebo jiným programem prolomit tyto hashe (brute force + slovník). Ukaždého uveďte jakým programem a jak dlouho to trvalo:

```
5f4dcc3b5aa765d61d8327deb882cf99
e99a18c428cb38d5f260853678922e03
8d3533d75ae2c3966d7e0d4fcc69216b
0d107d09f5bbe40cade3de5c71e9e9b7
5f4dcc3b5aa765d61d8327deb882cf99
```

Vypracujte tyto otázky:

1) Jakou spojitost má těžba kryptoměn a prolamování hesel?
2) Co a jak ovlivňuje délku a složitost prolamování. Popište a ukažte vzorový výpočet
3) Jak zajistit nejvyšší bezpečnost pro přihlášení k nějaké službě? (Popis jak, proč, atd)
4) Proč nestačí dělat programy na prolamování hashe k získání hesla?
5) Jak postupovat pokud chci udělat BruteForce útok na zaheslovaný zip soubor a jak postupovat, pokud chci udělat BruteForce útok na aplikaci, který vyžaduje heslo. (Popis vlastního řešení -> Jak dostat zkoušené hesla do formuláře jiné aplikace)

BONUS

1) Popište jak se implementuje přihlášování pomoci biometrických dat (otisk, obličej, duhovka -> včetně popisu ukládání a jak se řeší odchylky v těchto datech)
2) Jak prolamovat zařízení / aplikace s ochranou proti BruteForce (celý postup -> automatizovaný)

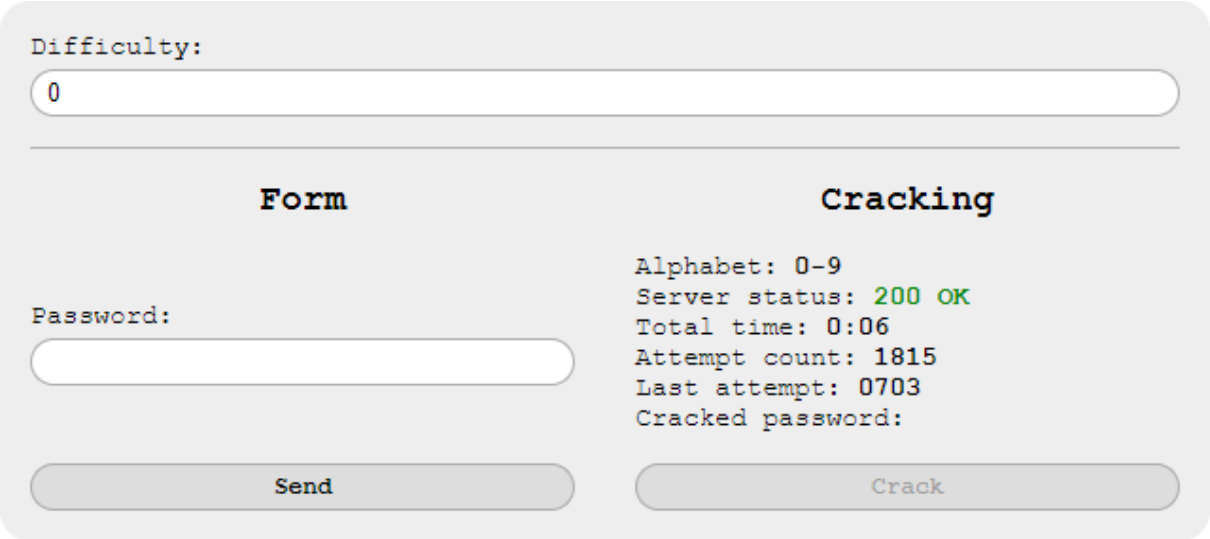
Vypracování

Příprava webové aplikace

V rámci tohoto protokolu bylo za úkol vytvořit webovou aplikaci s formulářem. Tento formulář obsahuje volbu složitosti od 0 do 4 a je implementován jako element *input[type="number"]*. Tento element ovlivňuje, jaké heslo se má na straně serveru porovnávat s heslem odeslaným ve formuláři, případně v rámci prolamování pomocí JavaScriptu, a dále také rozsah znaků (abeceda) použitých pro generování všech možných variací.

Po zadání hesla (lze odeslat i prázdné heslo) a kliknutí na tlačítko „Send" formulář automaticky přesměrovává na stránku *server.php*, která nejprve zkontroluje, zda byly odeslány požadované údaje (složitost a heslo) a vypisuje, zda se jedná o správné heslo (vypíše: „OK") nebo o špatné heslo (vypíše: „KO"). V případě, že nebyla odeslána složitost či heslo vypíše chybovou hlášku „400 Bad Request". Pokud je zapnutá ochrana proti BruteForce, tak ještě vypisuje hlášku „429 Too Many Requests" a to pouze když je u dané složitosti aktuálně naplněn max. počet pokusů a poslední nesprávný pokus byl odeslán před méně než 5 minutami.

Pro prolamování jsou dále na webu statistiky, které se aktualizují vždy jednou za sekundu, aby příliš nesnižovaly výkon prolamování. Položka „Server status" zobrazuje status HTTP požadavku, který je při vypnuté ochraně většinou „200 OK". Při zapnuté ochraně pak po vyčerpání limitu zobrazuje „429 Too Many Requests". „Total time" je aktuální celkový čas, který byl dosud vynaložen na prolamování a „Attempt count" je pak počet pokusů neboli variací řetězců, které byly odeslány na server. „Last attempt" obsahuje poslední řetězec odeslaný na server a „Cracked password" je prolomené heslo, které se zobrazí až po úspěšném prolomení. Obdobná statistika vypisovaná jednou za sekundu je také ve vytvořené aplikaci v Pythonu.



Obr. 1 - Ukázka webové aplikace

Prolamování hesel – JavaScript vs Python

Pro generování všech možných kombinací nebyla ani v JavaScriptu ani v Pythonu využita knihovna, aby byl časový výsledek porovnatelný nad stejným počtem pokusů.

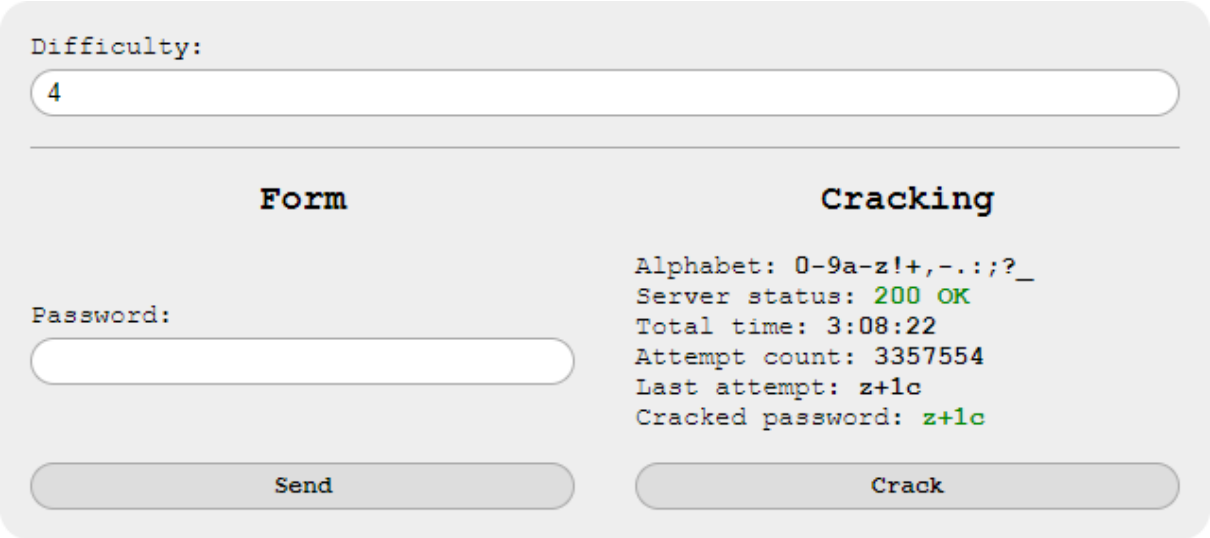
Prolamování v JavaScriptu i v Pythonu bylo prováděno na těchto specifikacích:

- Procesor: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
- Grafická karta: Intel(R) HD Graphics 630 / NVIDIA GeForce GTX 1050
- Paměť: 8.00GB
- Disk: SanDisk SD8SN8U128G1002
- Operační systém: Windows 10 Home (verze: 22H2, build: 19045.2193, typ: x64)

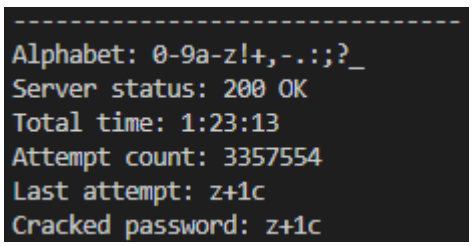
Webová aplikace, ve které bylo spouštěno prolamování pomocí JavaScriptu byla spuštěna v prohlížeči Microsoft Edge (verze: 107.0.1418.26, 64bitová).

Heslo	Rozsah znaků pro generování	Počet pokusů	Čas	
			JavaScript	Python
5612	0-9	6 724	0:23	0:10
32276	0-9	43 388	2:24	1:04
bzcd	a-z	52 811	3:02	1:17
h4b2	0-9a-z	846 724	47:54	21:35
z+1c	0-9a-z!+,-.:;?_	3 357 554	3:08:22	1:23:13

Tab. 1 - Statistika prolamování hesel



Obr. 2 - Ukázka prolamování hesla o složitosti 4 (JavaScript)



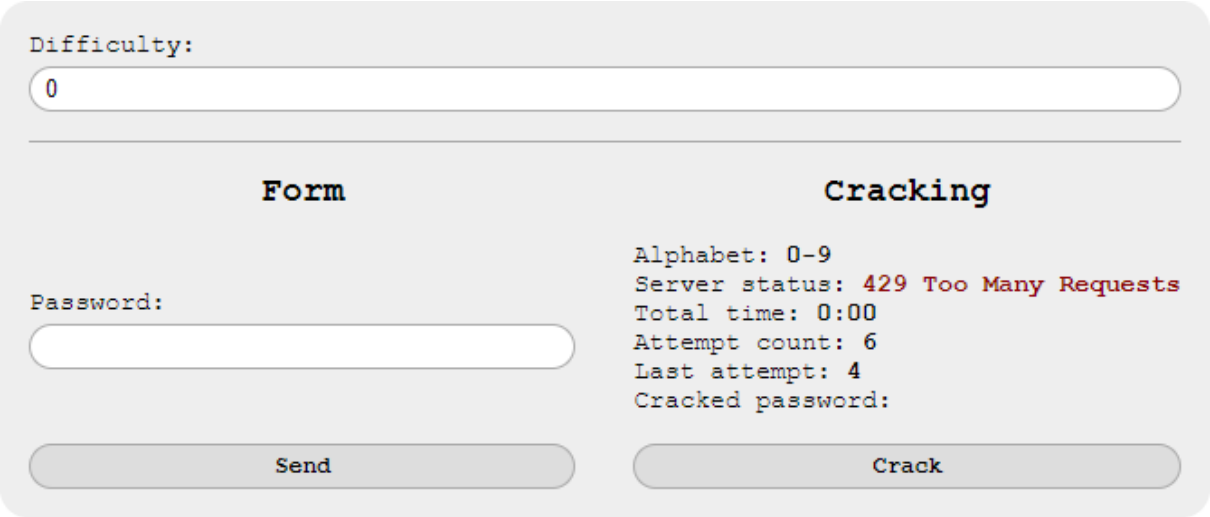
Obr. 3 - Ukázka prolamování hesla o složitosti 4 (Python)

Jak je možné vidět na tabulce uvedené výše, Python všechny hesla prolomil mnohem rychleji než JavaScript. Toto chování je způsobeno tím, že JavaScript běží v prohlížeči, a tak musí v rámci požadavku řešit také spoustu dalších kontrol jako cross origin, bezpečnost, cookies a další. Tyto kontroly pak výrazně snižují celkovou rychlost požadavku a jeho odpověď dorazí tedy o něco později než u Pythonu, který tyto kontroly dělat nemusí.

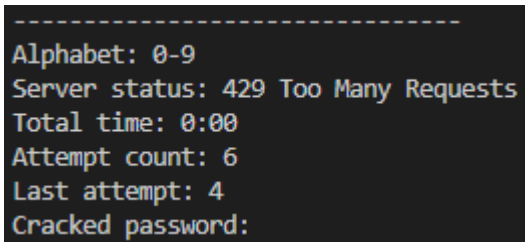
Rozsah znaků pro generování byl omezen u písmen záměrně pouze na malé písmena a u speciálních symbolů pouze na vybraných 9 z 33 možných z důvodu časové náročnosti generování.

Ochrana proti BruteForce

Ochrana proti BruteForce funguje tak, že si na serveru u jednotlivých složitostí uchovává počet špatných pokusů až do naplnění max. limitu 5 pokusů a také čas posledního špatného pokusu. Tyto data si pak ukládá do souboru ve formátu JSON. Jakmile je pro danou složitost naplněn limit, tak dokud aktuální čas pokusu není od posledního špatného pokusu více než 5 minut, pak vždy vrátí odpověď „429 Too Many Requests“ místo běžné odpovědi „OK“ či „KO“ a to i za předpokladu, že je dané heslo správné. Není tedy možné zjistit správnost více než 5 pokusů v rámci 5 minut. Po 5 minutách od posledního špatného pokusu u dané složitosti je pak při prvním pokusu předchozí počet špatných pokusů vyresetován, takže je opět možné provést 5 pokusů před dosažením limitu. Danou ochranu je nutné zapnout v souboru *server.php* nastavením proměnné *$bruteForceProtection* na *true*.



Obr. 4 - Ukázka funkční ochrany proti BruteForce (JavaScript)



Obr. 5 - Ukázka funkční ochrany proti BruteForce (Python)

Prolamování hashů – BruteForce + slovník

Prolamování hashů bylo provedeno na stejném HW jako prolamování hesel a byl k tomu využit nástroj **Hashcat (verze: 6.2.6)**, který umožňuje prolamování buď čistě pomocí BruteForce či BruteForce s využitím slovníku. Pro porovnání bylo provedeno prolamování pomocí obou metod a jako slovník byl využit **rockyou.txt**, který je dostupný na webu https://wiki.skullsecurity.org/index.php/Passwords.

Toto prolamování funguje tak, že se pomocí hrubé síly generují řetězce, které jsou následně zahashovány pomocí algoritmu MD5 a jsou porovnávány s daným hashem, dokud nedojde k jejich shodě. V případě shody je pak daný vygenerovaný řetězec, jehož hash se shodoval, tím heslem, které jsme se snažili získat.

Vzhledem k časové i finanční náročnosti generování náhodných řetězců se dnes mnohem častěji používá slovníkový útok, který vychází z toho, že uživatelé dost často používají velmi jednoduché a často opakující se hesla. Prioritně tedy nejprve zkouší hesla jako „password“, „1234“ a další, jelikož je u nich vyšší pravděpodobnost, že budou nalezeny dříve než různé vygenerované řetězce. Jak je možné vidět na tabulce níže, využití slovníku opravdu výrazně snížilo čas potřebný k nalezení daných řetězců.

Postup k prolomení hashů pomocí BruteForce:

1) Vytvořit soubor *hash.txt* a naplnit jej hashy (vždy jeden hash na řádek)
2) Otevřít PowerShell a nasměrovat jej do aktuální složky
3) Dále zadat následující příkaz: *.\hashcat -a 3 -m 0 -O hash.txt*

Postup k prolomení hashů pomocí slovníkového útoku:

1) Vytvořit soubor *hash.txt* a naplnit jej hashy (vždy jeden hash na řádek)
2) Otevřít PowerShell a nasměrovat jej do aktuální složky
3) Dále zadat následující příkaz: *.\hashcat -a 0 -m 0 -O hash.txt rockyou.txt*

Jednotlivé hashe a řetězce, které jsou v nich ukryté se poté postupně zobrazí.

Hash	Řetězec	Čas	
		BruteForce	Slovníkový útok
5f4dcc3b5aa765d61d8327deb882cf99	password	1:22	0:05
e99a18c428cb38d5f260853678922e03	abc123	0:44	0:05
8d3533d75ae2c3966d7e0d4fcc69216b	charley	0:45	0:05
0d107d09f5bbe40cade3de5c71e9e9b7	letmein	0:47	0:05

Tab. 2 - Statistika prolamování hashů

Závěr

1) Jakou spojitost má těžba kryptoměn a prolamování hesel?

Těžba kryptoměn má některé věci s prolamováním hesel společné. Těžba probíhá tak, že se těžař snaží vygenerovat takovou nonce, která společně s dalšími informacemi o bloku bude po zahashování vést k určitému počtu úvodních nul ve výsledném hashi. Prolamování hesel je pak o snaze vygenerovat určitý řetězec, který bude po zahashování vést ke stejnému hashi, který se snažíme prolomit. V obou případech tedy máme generování na začátku a nějaké pravidla pro výsledný hash, ať už úplná shoda u prolamování hesel či shoda v úvodním počtu nul u těžby. U Bitcoinu se pro zahashování používá algoritmus SHA256, který se dnes rovněž používá pro hashování hesel.

2) Co a jak ovlivňuje délku a složitost prolamování. Popište a ukažte vzorový výpočet

Délku a složitost prolamování ovlivňuje síla hesla, tedy jeho délka, kombinace čísel, velkých a malých písmen a speciálních symbolů. Počet všech možných kombinací k prolomení daného hesla je pak $počet\ povolených\ znaků^{délka\ hesla}$, což pro heslo o délce 4 s povolenými znaky – číslice (10), velká a malá písmena anglické abecedy (52) dává celkem 62^4 možných kombinací.

Sílu hesla pak dále ovlivňuje také jeho unikátnost tedy, zda dané heslo neobsahuje např. po sobě jdoucí čísla či znaky. Heslo by také nemělo být velmi jednoduché či často používané, jelikož by mohlo být mnohem rychleji uhodnuté pomocí slovníkového útoku.

3) Jak zajistit nejvyšší bezpečnost pro přihlášení k nějaké službě? (Popis jak, proč, atd)

Pro zajištění nejvyšší bezpečnosti pro přihlášení je kromě silného hesla potřeba využít dalších metod ověření, jelikož samotné heslo může být postupem času ze systému uhodnuto či může být také součástí např. úniku databáze.

Jedním ze způsobů může být využití nějaké kontrolní otázky např. jméno psa či příjmení matky za svobodna. Tato otázka do určité míry snižuje riziko neoprávněného přihlášení, jelikož se může přihlásit pouze uživatel, který zná tyto osobní informace. V dnešní době sociálních sítí však nemusí být velký problém tyto informace odhalit zvlášť pokud se jedná o útok cílený na danou osobu.

Značně bezpečnější je využití dvoufaktorového ověření ať už pomocí SMS či s využitím aplikace (Google Authenticator, Authy). Pro přihlášení je tedy nutné mít fyzický přístup k zařízení. U tohoto způsobu záleží také na zabezpečení samotného zařízení využitého při 2FA, jelikož toto zařízení může být odcizeno. Minimálně by dané zařízení mělo být chráněno heslem či PINem, ideálně pak třeba otiskem prstu či duhovky. U SMS je pak ještě další problém s tím, že je potenciálně možné odchytit danou zprávu přímo ze vzduchu, kde by však šlo zneužití ztížit koncovým šifrováním.

Celkově je pak nejvyšší bezpečnost možné zajistit kombinací všech uvedených možností a to silným heslem, kontrolní otázkou, dvoufaktorovým ověřením pomocí aplikace (Google Authenticator, Authy) na zařízení zabezpečeném pomocí otisku prstu.

4) Proč nestačí dělat programy na prolamování hashe k získání hesla?

Jelikož program na prolamování hashe nám nezajistí získání samotného hashe. Hash je možné získat např. pomocí SQL injection, kdy je zneužita zranitelnost systému, která umožňuje číst libovolná data z databáze.

Mnohem jednodušší je pak získat heslo přímo od uživatele např. pomocí podvržení falešné webové stránky, infikováním jeho zařízení pomocí přílohy spamového emailu či falešným telefonátem pod jménem důvěryhodné instituce.

Dále pak může být výhodné zjišťovat informace o dané osobě např. ze sociálních sítí, jelikož by tyto informace mohly být využity pro vhodný slovníkový útok. Uživatel tedy nemusí mít heslo, které by bylo obecně snadno nalezené pomocí obecného slovníkového útoku, ale může být snadno nalezeno pomocí slovníkového útoku, který bude obsahovat osobní informace uživatele.

5) Jak postupovat, pokud chci dělat BruteForce útok na zaheslovaný zip soubor a jak postupovat, pokud chci udělat BruteForce útok na aplikaci, která vyžaduje heslo. (Popis vlastního řešení -> Jak dostat zkoušená hesla do formuláře jiné aplikace)

Pro BruteForce útok na zaheslovaný zip soubor je nejprve potřeba najít způsob, jak tento soubor inicializovat (v Pythonu na to existuje knihovna *zipfile*). Poté je třeba generovat všechny možné kombinace řetězců, které by mohly být potenciálně správným heslem. K tomu je možné využít již zmíněný slovníkový útok, který má vyšší pravděpodobnost na menší časovou náročnost generování, a tedy rychlejší prolomení.

Pro prolomení formuláře jiné aplikace je možné využít např. Selenium, které umožňuje spouštět WebDriver do kterého je možné vkládat libovolný vstup.

6) Popište, jak se implementuje přihlášování pomoci biometrických dat (otisk, obličej, duhovka -> včetně popisu ukládání a jak se řeší odchylky v těchto datech)

Otisk prstu, sken obličeje a duhovky porovnávají velké množství parametrů a vzorů, které jsou u každého jedince jiné. U otisku prstu se třeba porovnává počet vroubků kůže, jejich tvar, hloubka či délka. Otisk se i u stejného člověka může mírně lišit vzhledem k opotřebení kůže např. věkem, prací či chemikáliemi, takže kontrola počítá s určitou mírou přijatelné odchylky.

V mobilních zařízeních se u skenu obličeje často pracuje pouze s obrazem z kamery, který je možné podvrhnout např. fotkou dané osoby. Tato kontrola pak také velmi špatně funguje při nižším osvětlení obličeje. Vhodnější metody skenu obličeje pak používají další senzory např. s využitím infračerveného záření, které nejsou závislé čistě na obrazu a ty jsou jednak bezpečnější, tak také spolehlivější.

7) Jak prolamovat zařízení / aplikace s ochranou proti BruteForce (celý postup -> automatizovaný)

Ochrana proti BruteForce na zařízení obvykle implementuje počet nesprávných pokusů o přihlášení a v případě naplnění určitého limitu přihlášení na tento účet dočasně či úplně zablokuje. Tuto ochranu lze obejít tak, že si vytvoříme obraz paměti daného zařízení a nad jeho kopií postupně zkoušíme různé kombinace, dokud se daný obraz nezablokuje. Při

zablokování pak použijeme další kopii, a tak postupujeme, dokud nedojdeme ke správnému heslu.

Weby pak často pracují s ochranou pomocí CAPTCHA. K prolomení této ochrany dnes existuje několik služeb a knihoven, které využívají hlubokého učení pro řešení těchto úloh. Mezi tyto služby patří např. GRIS, Alchemy, Clarifai či NeuralTalk.

Proti BruteForce se lze ještě bránit blokací konkrétní IP adresy spojené s větším množstvím špatných pokusů o přihlášení. Tuto ochranu lze obejít např. využitím VPN či sítě TOR.