# بررسی چالش Vote Flooding:

## تعریف مسئله:

یک وبسایت article hosting داریم که در آن کاربران می‌توانند article ها را بخوانند و به هر کدام یک امتیاز از ۰ تا ۵ بدهند. چالشی که داریم این است که از طریق واسطه‌هایی، تعداد زیادی کاربر ممکن است تشویق به رای بالا/پایین دادن به یک article خاص شوند. مثلا ممکن است یک کانال تلگرامی محبوب به کاربرانش بگوید که یک article خاص را up vote یا down vote کنند. در این چالش سعی داریم این نوع «حمله» ها را شناسایی و خنثی کنیم.

## رویکرد درگیری با چالش:

**۱. بررسی موارد مشابه:**
ابتدا سعی می‌کنیم در وبسایت‌های بزرگ و معروف، چالش‌های مشابه و نحوه حل آنها را پیدا و بررسی کنیم و سعی کنیم از راه‌حل‌های آنها الهام بگیریم یا حتی همان راه‌حل‌ها را با تغییراتی به نیاز خودمان fit کرده و استفاده کنیم.

**۲. بررسی داده‌های حمله‌های قبلی:**
در یک سناریوی واقعی، احتمالا این مدل حمله قبلا اتفاق افتاده است و dataی آن را خواهیم داشت. این data برای ساخت راه‌حل بسیار ارزشمند خواهد بود چرا که الگوها و insightهای ارزشمندی را در اختیارمان خواهد گذاشت. مثلا می‌توانیم هر راه‌حل را بر اساس اینکه چقدر این حملات قبلی را می‌توانست خنثی کند ارزیابی کنیم.
در همین راستا می‌توانیم طبق الگوها و insight های حملات گذشته راه‌حلی را طراحی کنیم که جلوی آنها را بگیرد.
در وضعیت فعلی دادهٔ واقعی نداریم ولی سعی خواهیم کرد چند insight معقول را فرض کنیم و طبق آنها عمل کنیم.

**۳. بررسی هزینه‌های هر راه‌حل پیشنهادی:**
برای هر راه‌حل، ابتدا هزینه‌های اعمال آن را بررسی خواهیم کرد. این هزینه‌ها در ابعاد مختلفی اعمال می‌شوند، از جمله:

- هزینه فنی پیاده‌سازی: برای اعمال هر راه‌حل نیاز به اضافه کردن functionality به سرویسمان خواهد بود و بالطبع راه‌حل‌های مختلف هزینه فنی پیاده‌سازی متفاوتی خواهند داشت: مثلا هزینه فنی یک ML Model پیچیده بسیار بیشتر از اعمال یک rule یا throttling ساده است.
- هزینه اضافه کردن پیچیدگی به سیستم: با اعمال روش‌های پیچیده جلوگیری از حمله به سیستم‌مان، داریم به system complexity اضافه می‌کنیم و business logic ای که در کد پیاده‌سازی شده‌است را بیشتر می‌کنیم. این موضوع باعث سخت‌تر شدن maintenance ،troubleshooting و extend کردن سیستم خواهد شد.
- هزینه UX: راه‌حل‌های تشخیص حمله ما ممکن است سناریوهای False Positive ایجاد کنند که باعث شود rating های سالم هم اعمال نشده و دور ریخته شوند که این موضوع تاثیر منفی روی UX خواهد داشت. همچنین برخی راه‌حل‌هایی که کاربران مشکوک به حمله را به نحوی تنبیه میکنند (مثلا اکانت آنها را ban می‌کنند) در صورت بروز false positive هزینه UX بالایی اعمال می‌کنند.

**۴. بررسی اثرگذاری راه‌حل‌های پیشنهادی خودمان:**

بعد از بدست آوردن چند راه‌حل و بررسی هزینه‌های هر کدام، باید هر راه‌حل را از منظر اثرگذاری بررسی کنیم. به این معنی که هر راه‌حل چند درصد از حملات را شناسایی و خنثی خواهد کرد. در این بررسی باید حواسمان به سناریوهای False Negative باشد (سناریوهای False Positive در قسمت بررسی هزینه کاور شده‌اند.)
همچنین در بررسی نفوذپذیری هر راه‌حل، باید فرضیاتی برای میزان knowledge اتکرها نسبت به سیستم‌مان، میزان منابع در اختیار اتکرها، و مواردی از این دست درنظر بگیریم. حالت adversarial حالتی را توصیف می‌کند که فرض می‌کنیم اتکرها کاملا سیستم‌مان را می‌شناسند و هر منبعی را در اختیار دارند، یعنی راه‌حل را در بدبینانه‌ترین حالت ممکن بررسی می‌کنیم. اثرگذاری در حالت adversarial نیاز به راه‌حل‌های خیلی سخت‌گیرانه خواهد داشت که هزینه‌های راه‌حل را زیاد خواهند کرد. پس نیاز داریم فرضیات معقولی را در نظر بگیریم و طبق آنها نفوذپذیری را ارزیابی کنیم.

**5. انتخاب بهترین راه‌حل با توجه به فاکتورهای مورد بررسی و منابع در اختیار:**

در این مرحله از نتایج بررسی‌مان استفاده خواهیم کرد و با توجه به محدودیت‌های منابعمان (بعنوان مثال محدودیت شدید نیروی انسانی) یک راه‌حل را انتخاب و پیاده‌سازی خواهیم کرد.

**حالا هر یک از این مراحل را انجام داده و نتایج آن را مستند می‌کنیم.**

# بررسی موارد مشابه:

بعد از بررسی چندین پلتفرم که در آنها user review موضوعیت دارد، راه‌حل‌های کلی زیر پیدا شد که هرکدام را توضیح خواهیم داد.

## Rate Limiting

**شرح راه‌حل:**
در این مورد برای هر کاربر یک سقف rate برای انجام یک action مثل ثبت امتیاز تعریف می‌شود. مثلا در reddit کاربری نمی‌تواند روزانه بیشتر از ۱۰ بار پستی را vote کند. البته مقدار این سقف طبق سوابق کاربر (user reputation که در ادامه شرح داده شده‌است) تعیین می‌شود. Telegram و Instagram هم از روش مشابه استفاده می‌کنند ولی نه برای review بلکه برای تعداد follow ها یا subscription ها در روز.
این راه‌حل جلوی اکانت‌های «اجاره‌ای» که طبق دستور کسی بصورت دسته‌جمعی action ای مثل عضو شدن در یک کانال را انجام می‌دهند را می‌گیرد. نمونه‌ای از استفاده این اکانت‌ها در اپ‌های «فالوور جمع‌کن» و پلتفرم‌های مشابه دیده می‌شود. مثلا کاربران این اپ مجبورند روزانه تعداد زیادی اکانت را فالو کنند که rate limit جلوی آنها را خواهد گرفت.

**پلتفرم‌هایی که از این راه‌حل استفاده می‌کنند:**
پلتفرم‌های بزرگی مثل Telegram, Instagram, Reddit, Twitter و بسیاری دیگر از فرم‌های متفاوتی از rate limiting استفاده می‌کنند.

**بررسی portability راه‌حل:**
این روش به این خاطر که rule based و ساده‌است با هزینه بسیار کمی می‌تواند در domain ما پیاده‌سازی شود.
همچنین می‌توان علاوه بر per user rate limiting به per article rate limiting هم فکر کرد. اما محدود کردن امتیازدهی‌های یک article عواقب زیادی دارد. مثلا بسیار محتمل است یک article واقعا انقدر جنجالی باشد که بصورت طبیعی یک rating surge ایجاد کند و با بلاک کردن آن جلوی flow سالم را گرفته باشیم. در این راستا می‌بینیم که تقریبا هیچ پلتفرمی هم کاری مشابه این را نکرده است و در تمامی موارد فقط per user rate limiting داریم.
البته می‌توان از per article rate surge بعنوان یک نشانه‌ی حمله استفاده کرد و این الگو را برای بررسی دستی flag کرد (کاری که پلتفرم‌های مثل youtube انجام می‌دهند.)

## User Reputation

**شرح راه‌حل:**

جهت جلوگیری از ratingهای هرز (و تعدادی آسیب‌پذیری دیگر)، یک **سیستم user reputation** بسیار پیاده می‌شود. منطق این سیستم به این صورت است که هر کاربر یک reputation score دارد که هنگام ثبت‌نام ۰ است. هرچه کاربر فعالیت و سابقه بیشتری در وبسایت ثبت کند (مثلا در Stack Overflow سوالاتی بپرسد یا جواب‌هایی بدهد که rating بالایی میگیرند، یا در AirBnb تعداد دفعات بیشتری خانه اجاره کنند یا اجاره بدهند)، reputation بیشتری دریافت می‌کند.

اثرگذاری rating های یک user کاملا وابسته به reputation او خواهد بود. مثلا در بازه‌های پایین reputation کاربر حق rating نخواهد داشت. با افزایش reputation امکان نظردهی فراهم می‌شود و در بازه‌های بالاتر نظرات کاربر مهم‌تر و اثرگذار تر خواهد بود.

برخی از این این سیستم این ویژگی را هم دارد که کاربران با reputation بالاتر تصمیم می‌گیرند کدام user های با reputation پایین‌تر promote شوند. مثلا اگر کاربری با رنک بالا یک پست یک کاربر با رنک پایین تر را up vote/down vote کند، تاثیر مثبت یا منفی زیادی روی reputation او خواهد داشت.

**پلتفرم‌هایی که از این راه‌حل استفاده می‌کنند:**

این راه‌حل بسیار فراگیر است و پیاده‌سازی‌های آن در AirBnB، Quora، Amazon، Reddit، Stack Overflow و ... دیده می‌شود.

**Amazon:**

در product review های این وبسایت ecommerce، کامنت‌های کاربران مقبول‌تر ابتدا نمایش داده می‌شود و به rating آن‌ها وزن‌دهی بالاتری می‌شود. همچنین کاربرانی که نظرات قبلی‌شان کارآمد بوده طی روال‌هایی به‌عنوان Top Reviewer شناخته می‌شوند و کامنت‌هایشان بالاتر و با همین لیبل نشان داده می‌شود. علاوه بر این rating و review های کاربرانی که کالا را خریداری کرده‌اند وزن بیشتری دارد.

**AirBnB:**

کاربرانی که تعداد دفعات بیشتری host یا guest بوده‌اند وزن بیشتری برای rating دارند.

پیاده‌سازی این روش در دیگر پلتفرم‌ها مشابه است و وارد جزییات آنها نخواهیم شد.

**بررسی portability راه‌حل:**

برای اینکه بتوانیم در domain امان این سیستم را پیاده کنیم، باید کاربران بتوانند action های انجام دهند که طبق آن بتوان به آنها امتیاز داد. از آنجایی که functionality وبسایت ما به شدت محدود است، چنین action هایی در این مرحله به تعداد زیادی وجود ندارند.

ویژگی‌هایی که می‌توان طبق آن کاربران دامین ما را rank کرد:

- قدمت اکانت: اکانت‌های قدیمی‌تر مقبول تر باشند. مثلا در ماه اول ثبت‌نام کاربر نتواند rating ثبت کند. یا هر کاربر یک وزن برای rating خود داشته باشد که در ابتدا از ۰ شروع می‌شود و به مرور زمان زیاد می‌شود.
- اضافه کردن موانع احراز هویتی یا pay wall: می‌توان مقبولیت بیشتر را وابسته به مراحل دست‌وپا گیری کرد که باعث شود کاربرانی که صرفا می‌خواهند یک بار وارد شده و یک article را up vote یا down vote کنند ریزش داشته باشند. مثلا می‌توان مرحله احراز هویت از طریق sms/email validation یا حتی استعلام کد ملی، درخواست آپلود تصویر مدارک و ... اضافه کرد. البته این موارد پیچیدگی‌های حقوقی‌ای اضافه می‌کنند که بررسی نخواهیم کرد. Pay Wall و فروش اکانت premium که مقبول تر است هم راه خوبی برای تمایز بین کاربران «جدی» و غیر است.

حس می‌شود که معیارهایی که برای ranking کاربران داریم محدود هستند و این راه‌حل شاید بصورت مستقیم قابلیت port شدن به دامین ما را نداشته باشد.

## Community Moderation

**شرح راه‌حل:**

ساده‌تر است که این روش را در کانتکست پیاده‌سازی آن در Reddit توضیح دهیم. در این پلتفرم تعداد زیادی subreddit فعالیت می‌کنند که توسط کاربرانی moderate می‌شوند (هر subreddit تعدادی moderator دارد که از کاربران عادی هستند.) این moderator ها توانایی دارند کاربران خراب‌کار را طبق صلاحدید خودشان از subreddit خود ban کنند یا دسترسی آنها را محدود کنند (مثلا حق دادن امتیاز را از آن‌ها بگیرند.)

**بررسی portability راه‌حل:**

مسئله اصلی port کردن این راه‌حل به دامین ما این است که افرادی برای moderate کردن نداریم. تنها کاندیدا این است که نویسنده هر article را مسئول moderation همان article کنیم ولی در این صورت این فرد incentive دارد که کاربرانی که rating کمی میدهند را تنبیه یا به نحو دیگری خنثی کند و این مسئله feedback mechanism سایت ما را مختل خواهد کرد. در غیاب نویسنده، بدون عوض کردن گسترده دومین (مثلا معرفی چیزی مثل subreddit ها) نمی‌توانیم این راه‌حل را پیاده سازی کنیم. از آنجایی که برای رفع یک مشکل امنیتی نباید ذات محصولمان را عوض کنیم، تصمیم می‌گیریم که این راه‌حل برای دومین ما مناسب نیست.

نوع دیگری از moderation که در آن admin هایی که تضاد منافع ندارند (مثلا کارمند خود ما هستند) وظیفه moderation را برعهده می‌گیرد در صورتی که سازمان ظرفیت نیروی انسانی مناسب را داشته باشد قابل انجام است. می‌توانیم با استفاده از سیستم‌ها و قوانینی فعالیت‌های مشکوک را flag کنیم تا moderator هایمان آنها را بررسی کنند.

## Behavioral Analysis and Anomaly Detection

**شرح راه‌حل:**

در این روش توسط مدل‌های ML رفتار کاربران (هم به صورت تکی و هم به صورت جمعی) بررسی شده و الگوهای نرمال آن پیدا می‌شوند. سپس در صورتی که الگوی فعالیت فعلی با الگوی کلی تضاد پیدا کند، یک alert اعلام می‌شود و بررسی‌های بیشتری صورت می‌گیرند. استراتژی دیگر برای موارد حساس‌تر این است که در صورت رخداد فعالیت خارج از الگوی نرمال action ها فریز شوند تا از حمله احتمالی جلوگیری شود. بعد از بررسی بیشتر یا فروکش حمله محدودیت‌ها برداشته می‌شوند.

**بررسی portability:**

باز هم بخاطر کوچک بودن دومین ما و همچنین محدودیت داده و منابع فنی، این راه‌حل برای ما مناسب نخواهد بود.

# بررسی داده‌ها:

همانطور که اشاره کردیم، در غیاب داده‌های واقعی، ناچاریم با شرایط معقولی تعدادی insight فرض کنیم و طبق آنها تصمیم‌گیری کنیم.
در تجمیع داده‌های اتک‌های قبلی، علاوه بر ثبت آنچه توسط سیستم ما مشاهده شد، باید دلیل حمله را هم داشته باشیم. مثلا بدانیم دلیل یک حمله این بوده که کانال x در تلگرام از اعضایش خواسته بوده که یک مقاله را down vote کنند.
در این بررسی فرض می‌کنیم با استفاده از Captcha و روش‌های دیگر جلوی حمله از طرف بات‌ها گرفته می‌شود و روی insight هایی که به شناسایی حملات از طرف کاربران واقعی انجام می‌گیرد تمرکز خواهیم کرد. همچنین فرض می‌شود که برای ثبت rating نیاز است کاربر login کرده باشد پس مسائل مربوط به پنهان‌کردن هویت از طریق تغییر ip و ... موضوعیت نخواهند داشت.

## Insights

**۱. امتیازات حدی در حمله‌ها:**

احتمالا کاربرانی که در قالب یک حمله امتیازدهی می‌کنند، امتیازات حدی می‌دهند. مثلا در صورتی که بخواهند یک مقاله را down vote کنند با امتیاز ۰ این کار را می‌کنند و اگر بخواهند up vote کنند با امتیاز ۵ انجام می‌دهند. انتظار داریم در ترافیک اتک امتیازات میانی مثل ۱ یا ۴ دیده نشوند.
این موضوع توسط بررسی داده‌های اتک‌های قبلی به سادگی قابل تصدیق است.
همچنین می‌توان بجای طرز فکر گسسته به موضوع (اینکه در حملات دقیقا امتیازات ۰ و ۵ زیاد می‌شوند) بصورت آماری نگاه کرد و ادعا کرد که Standard Deviation امتیازات موضوعیت بیشتری دارد. در حملات Standard Deviation امتیازات کمتر خواهد بود زیرا تعداد زیادی کاربر بصورت هم‌صدا up vote یا down vote می‌کنند. حدس میزنیم این روش آماری نسبت به روش گسسته و rule based نتیجه بهتری بدهد. البته با استفاده از داده‌ها می‌توانیم این دو منظر را benchmark و مقایسه کنیم.

**۲. کاربران جدید در حمله‌ها:**

انتظار داریم در حمله‌ها، تعداد زیادی از کاربرانی که برای انجام یک اکشن هدایت شده‌اند، از قبل اکانت نداشته بوده باشند یا اکانتشان در حالت عادی فعالیت زیادی نداشته بوده باشد. بخصوص در اتک‌هایی که عامل‌شان قبلا به سایت ما حمله نکرده باشد این موضوع مشهود خواهد بود. مثلا اگر مقاله‌ای راجع به یک سلبریتی نوشته شده باشد و یک fan page او از کاربرانش بخواهد مقاله را up vote کنند، احتمالا اکثر این کاربران از قبل اکانت نداشته‌اند.

**۳. کاربران یکسان در حملات با عوامل یکسان:**

در حملاتی که به خاطر دلیل یکسان انجام شده‌اند، انتظار داریم کاربرانی که action هرز انجام داده‌اند یکسان بمانند. مثلا کاربرانی که در حمله به یک مقاله درباره یک سلبریتی نقش داشته‌اند، احتمالا در حملات بعدی مربوط به او نیز مشارکت خواهند داشت و یک cluster را تشکیل می‌دهند.

**۴. الگوی زمانی در حملات:**

این مسئله تقریبا بدیهی است که حملات در بازه زمانی کوتاه و بصورت surge انجام خواهند شد پس یک حمله احتمالا باعث ثبت تعداد زیادی از نظرات به صورت ناگهانی خواهد شد.

البته مقالاتی که جنجالی هستند یا جنجالی می‌شوند هم ممکن است الگوهایی مشابه حمله در فعالیت‌ها ایجاد کنند و باید حواسمان به این شباهت باشد.

**۵. عدم همخوانی نرخ امتیاز گیری با انواع دیگر engagement با مقاله:**

اینکه یک کاربر به یک مقاله امتیاز می‌دهد یا خیر تنها یک مدل engagement با مقاله است. در دومین ما انواع دیگر engagement مثل کامنت گذاشتن یا به اشتراک‌گذاری وجود ندارد ولی می‌توانیم از «میزان زمانی که کاربر در صفحه مقاله مانده است» و اینکه «کاربر تا کجای صفحه مقاله را scroll down کرده است» بعنوان معیارهایی برای engagement استفاده کنیم.

حدس می‌زنیم در حملات نرخ امتیازگیری یک مقاله بطرز ناخوانایی با میزان engagement آن زیاد شود. یعنی کاربران بدون اینکه مقاله را بخوانند یا مدت زیادی روی آن بمانند امتیاز می‌دهند.

این موضوع هم به سادگی با استفاده از داده‌های واقعی قابل تصدیق است.

**۶. الگوی referrerهای مشابه در حملات:**

اگر عواملی باشند که وب‌سایت یا وبلاگ هستند و کاربرانشان را برای انجام یک action به سایت ما می‌فرستند، این کاربران با هدر referrer یکسان وارد وبسایت ما خواهند شد و از این طریق قابل تشخیص خواهند بود. البته این عوامل به سادگی می‌توانند با تعویض referrer policy خود جلوی این روش ما را بگیرند ولی قابل تصور است که بسیاری از آنها دانش فنی تشخیص و حل این موضوع را نداشته باشند.

**۷. نحوه ورود به صفحه مقاله در مقالات:**

در یک حمله، احتمالا کاربران با flow طبیعی (وارد شدن به لیست مقالات، گشتن، و ورود به یک مقاله) وارد یک مقاله نمی‌شوند و بصورت مستقیم از طریق یک لینک به آن وارد می‌شوند. البته در حالتی که یک مقاله زیاد اشتراک‌گذاری شود هم ممکن است بصورت ارگانیک ورودهای مستقیم به یک مقاله از طریق لینک زیاد باشند.

## راه‌حل ها:

**۱. کاربران در ۱۰ روز اول ثبت‌نام خود حق rate کردن مقالات را ندارند.** همچنین می‌توان این را اعمال کرد که کاربران تا وقتی ۱۰ مقاله را کامل نخوانده‌اند حق رای دادن نداشته باشند ولی تشخیص «خوانده شدن مقاله» بار فنی front end دارد و زمان و منابع پیاده‌سازی آن را نداریم.
**۲. هر کاربر در روز حق rate کردن بیشتر از ۱۰ مقاله را ندارد.** مقدار دقیق این limit می‌تواند با توجه به کاربران متفاوت شود ولی به دلایلی که بحث کردیم امکان راه‌اندازی user reputation را نداریم پس نمی‌توانیم این feature را اضافه کنیم.
**۳. هر کاربر قبل از ثبت امتیاز باید حداقل یک دقیقه روی صفحه‌ی مقاله مانده باشد.** این محدودیت باعث drop قابل توجهی در کاربرانی که فقط به منظور امتیازدهی وارد وبسایت شده‌اند می‌شود.
**۴. برای ثبت امتیاز نیاز به پر کردن captcha هست:** این موضوع باعث می‌شود حملات ربات‌ها خنثی شوند.

تعدادی راه‌حل وجود دارند که نیازمند تحلیل الگوها و کارهایی data هستند که منطق آنها را شرح خواهیم داد و داده‌های مورد نیاز آنها را store خواهیم کرد ولی قسمت‌های pattern detection را پیاده‌سازی نخواهیم کرد.
این روش‌های detection می‌توانند در دو حالت اجرا شوند:

1. **بصورت Real Time:** هرگاه الگوی مشکوکی ایجاد می‌شود، همان هنگام counter شود. این حالت بار فنی و پیچیدگی بیشتری خواهد داشت.
2. **بصورت دوره‌ای:** بعنوان مثال هر شب تعدادی DAG دیتایی اجرا شوند و الگوهای مشکوک به حمله را بررسی کنند و در صورت لزوم اثر آنها را reverse کنند

در دومین‌هایی که حملات باید درجا خنثی شوند (مثل توییتر یا وبسایت‌های خبری که طبق فیدبک کاربران محتوی را propagate می‌کنند)، هزینه پیاده‌سازی سیستم‌های تشخیص real time توجیه‌پذیر است اما در دومین ما از آنجایی که طبق rating ها exposure مقالات را تغییر نمیدهیم (مثلا سیستم recommender ای نداریم که مقالات محبوب را به کاربران بیشتری معرفی کند) لزومی به دادن هزینه بررسی Real Time نخواهد بود و کافیست مثلا هر شب DAG هایی را برای detection الگوها اجرا کنیم.

**۵. بررسی واریانس نمرات داده شده به هر مقاله:**
از آنجایی که ratingها را به همراه تاریخ ثبت‌شان ذخیره می‌کنیم، می‌توان واریانس نمراتی که هر مقاله در یک روز گرفته است را محاسبه و بررسی کرد. در صورت عجیب بودن مقدار آن، می‌توان بررسی کرد که آیا حمله‌ای رخ داده است یا خیر و اگر رخ داده باشد اثرات آن را خنثی کرد. البته باید حواسمان باشد ممکن است طی یک flow طبیعی یک مقاله واریانس نمرات کمی داشته باشد (مثلا مقاله‌ای راجع به گربه‌های بامزه احتمالا نمرات بالایی با واریانس کم دریافت کند.) و فقط روی حساب آلرت یکی از سیستم‌ها اثرات rating ها را خنثی نکنیم.

**۶. بررسی referrerهای امتیازدهندگان به هر مقاله:**

مطابق یکی از insight هایمان، referrer های امتیازدهندگان هر مقاله را بررسی می‌کنیم و اگر ترافیک زیادی از منبع یکسان refer شده بود، یک alert روشن می‌کنیم تا بررسی بیشتر و خنثی‌سازی انجام شود.

**۷. بررسی نرخ زمانی امتیازدهی به هر مقاله:**

این سیستم باید بررسی کند که اگر نرخ امتیازدهی به یک مقاله از حالت نرمال بیشتر شده بود و شاهد spike بودیم یک آلرت raise شود.

از آنجایی که rate surge یک فاکتور خیلی مهم در حملات است، می‌توانیم آن را به صورت rule based و real time هم بررسی کنیم. مثلا avg و std نرخ روزانه امتیازگیری هر مقاله را نگه داریم و در صورتی که در یک روز بیشتر از avg + 2 * std امتیاز دریافت شد، امتیازگیری آن مقاله را freeze کنیم (یعنی امتیازات جدید را ثبت کنیم ولی در میانگین rating دخالت ندهیم) تا بررسی‌های بیشتر انجام شود و یا حمله خنثی شود و یا مقاله unfreeze شود.

**۸. مانیتور کردن منابع محتمل برای trigger کردن حملات:**

در صورت وجود ظرفیت انسانی و فنی لازم، می‌توان حملات را پیشبینی کرد. مثلا کانال‌های تلگرامی یا وب‌سایت‌هایی که ممکن است باعث این حملات شوند را شناسایی و پایش کرد و قبل از انجام حمله فرآیند امتیازگیری مقاله مربوطه را freeze کرد.

قسمت‌هایی از این پایش می‌توانند automated باشند. مثلا روی پلتفرم‌هایی که مشکوک هستیم crawler هایی بگذاریم که در صورت مشاهده لینکی به مقاله‌ای در سایت ما alert میدهند. البته که هزینه crawl کردن قابل‌توجه است.

**۹. کلاستر کردن کاربران دارای مشارکت در حملات:**

طبق یکی از insightهایمان حملاتی که از سمت منبع یکسان آغاز می‌شوند غالبا شامل کاربران یکسانی هستند. از این رو می‌توان کاربران خراب‌کار را بر اساس داده‌های قبلی cluster کرد و روی این مورد alert های real time ست کرد. مثلا اگر بیشتر از ۵ درصد یک کلاستر خراب‌کار به یک مقاله نمره‌دهی کرده‌اند، یک alert برای حمله صادر شود. از آنجایی که از کاربران با سابقه خرابکاری برای تشخیص حملات آینده استفاده می‌کنیم، باید آنها را بصورت مخفیانه mark کنیم تا rating هایشان شمرده نشود ولی اکانتشان هم ban نشود تا تحت این تاثیر که rating هایشان موثر است به حملات ادامه دهند.

**۱۰. بررسی همخوانی نرخ امتیازدهی و دیگر انواع engagement:**

مطابق یکی از insight هایی که شرح دادیم، الگوهای عدم همخوانی نرخ امتیازگیری و دیگر انواع engagement روی هر مقاله می‌توانند شناسایی شوند و به عنوان alert برای حمله استفاده شوند.

در صورتی که بخواهیم داده‌ی انواع engagement کاربران را با هر مقاله ذخیره کنیم، راه‌حل دیتابیس مثل Postgres عملی نیست و باید از سولوشن‌های storage بزرگتری مثل HDFS استفاده کنیم.

**۱۱. Track کردن لینک‌های اشتراک‌گذاری:**

بسیاری از حملات به این صورت هستند که یک فرد لینک مقاله را از وبسایت ما بر می‌دارد و آن را در پلتفرمش (مثل کانال تلگرام) به اشتراک می‌گذارد و کاربران را دعوت می‌کند تا به آن وارد شوند. در این راه‌حل باید کار های زیر را انجام دهیم:

- در لینک مقاله، بصورت مستتر و در قالب یک query param شناسه‌ای بگذاریم که برای هر کاربر فرق کند. اینگونه هر لینک کپی شده مربوط به یک کاربر متفاوت خواهد بود و از روی لینک قابل تشخیص خواهد بود که کدام کاربر آن را به اشتراک گذاشته است.
- کاربرانی که با لینک یکسانی وارد شده‌اند به راحتی قابل تشخیص خواهند بود. حتی کاربری که لینک را به اشتراک گذاشته است هم معلوم خواهد بود.
- در صورت تشخیص یا شک به حمله، می‌توانیم بررسی کنیم چه میزان از rating ها توسط سورس یکسان وارد شده‌اند و آنها را خنثی کنیم.

ممکن است یک اتکر tech savvy متوجه این متد ما شود و سعی کند آن را مختل کند (مثلا شناسه را عوض کند.) می‌توانیم شناسه را بصورت sign شده وارد کنیم یا مثلا یک checksum خاص را در آن رعایت کنیم تا اتکر نتواند به راحتی شناسه جدید بسازد. همچنین می‌توانیم شناسه‌هایی که صادر کرده‌ایم را store کنیم و فقط اگر آنها داده شده بودند مقاله را سرو کنیم.

باید توجه کنیم که کاربرانی که بصورت طبیعی از طریق لیست مقالات وارد می‌شوند باید سرو شوند و به آنها یک شناسه جدید داده شود.

**نکته: بدلیل مضایقه در زمان، تمام این راه‌حل ها پیاده‌سازی نشدند.**