Radosław Wesołowski 66320

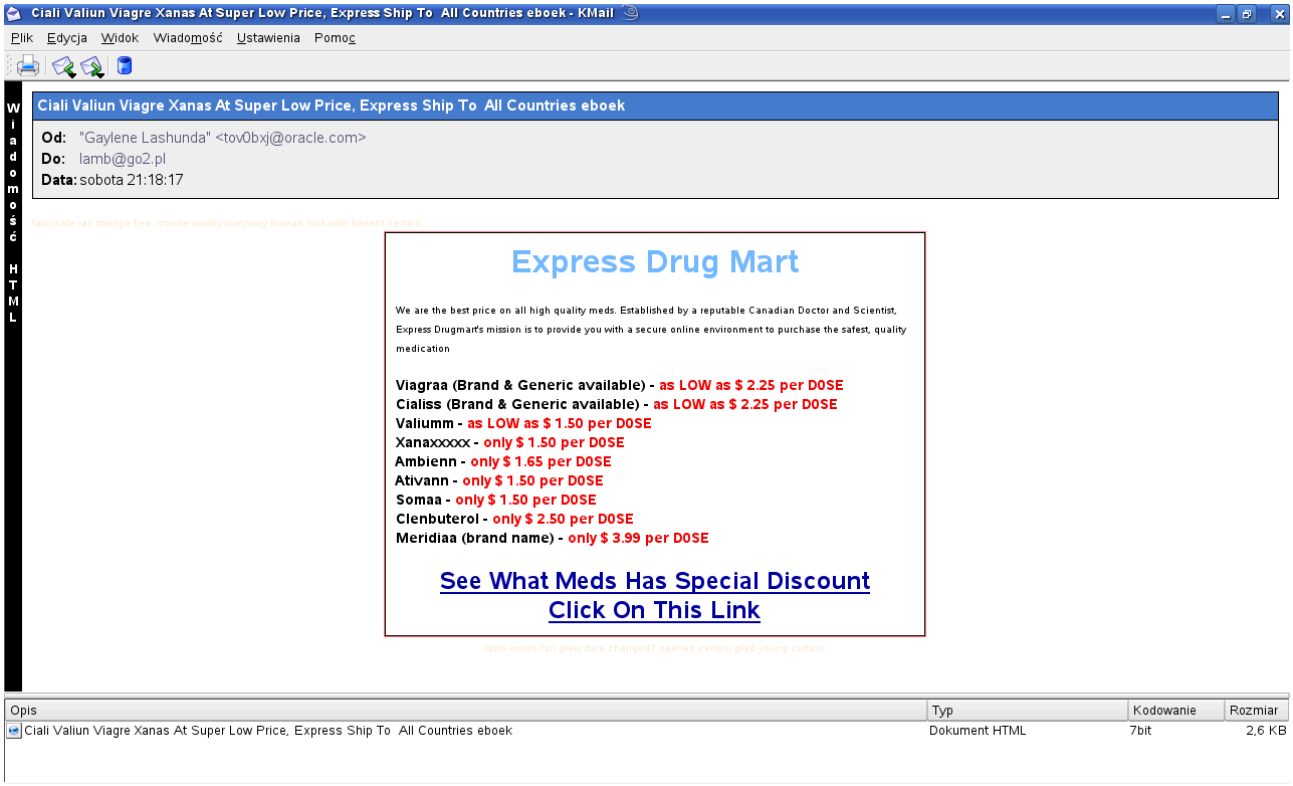
Rozpoznawanie obrazów – projekt zaliczeniowy

SpamKilla - wykrywanie spamu przesyłanego w postaci obrazków

Krótki opis projektu

Problem, którym zajmowałem się podczas trwania zajęć polegał na automatycznym wykrywaniu i klasyfikacji obrazkowego spamu przesyłanego w emailach. Technika ta polega na osadzaniu treści reklam w pliku HTML i przesyłaniu go do zwykłych użytkowników. Aby utrudnić wykrywanie takich przypadków spamerzy często zamieszczają część treści w postaci pliku graficznego. Standardowy klient pocztowy dokonuje interpretacji i ewaluacji kodu HTML, a dzięki specjalnie przygotowanym arkuszom styli CSS takie pliki są idealnie wkomponowane w całość i dla przeciętnego użytkownika wizualnie nie różnią się niczym od zwykłego tekstowego emaila.

Poniższy rysunek przedstawia przykładowy email będący spamem:



Obecnie problem spamu obrazkowego jest już dobrze znany specjalistom zajmującym zwalczaniem niechcianej poczty. Istnieje odpowiednie oprogramowanie działające po stronie serwerów pocztowych, które potrafi wykrywać i usuwać takie przesyłki. Dzięki wysiłkom dostawców usług internetowych w maju 2007 roku odsetek spamu obrazkowego wynosił zaledwie 15%. Dla porównania rok wcześniej było to aż 40%. Wykrywanie i analizowanie spamu obrazkowego jest

wbrew pozorom procesem dosyć prostym. Dzieje się tak, ponieważ informacja zawarta w pliku graficznym musi być przedstawiona w sposób czytelny, tak aby przeciętny użytkownik nie miał problemu z jej interpretacją. Taki stan rzeczy powoduje, że nawet bez specjalnych zabiegów poprawiających jakość obrazków, silniki OCR potrafią dosyć dobrze rozpoznać zawarty tekst i dokonać jego analizy na przykład sprawdzając, czy zawiera on słowa klucze znajdujące się na czarnej liście. Co więcej znaczna większość spamu posiada podobne zakłócenia (wprowadzane, aby utrudnić rozpoznawanie tekstu).

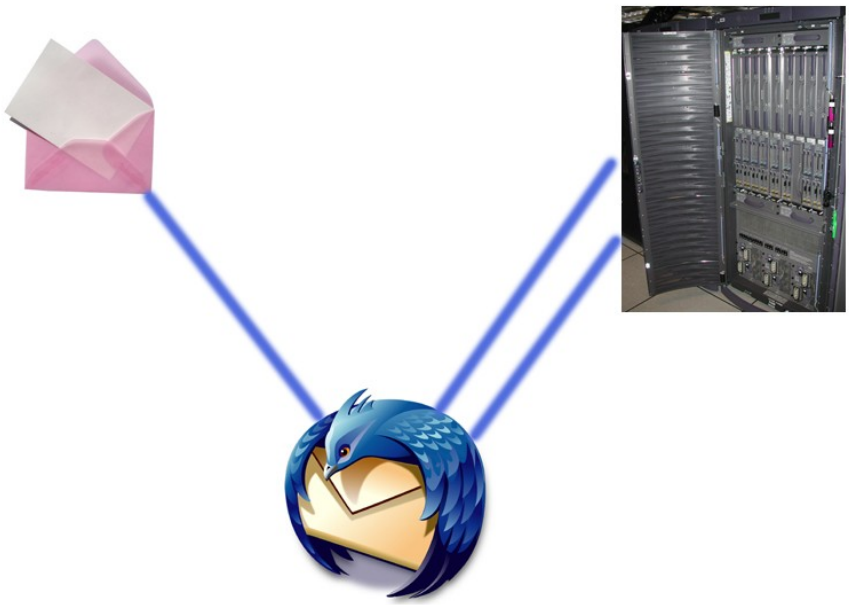
Architektura

Projekt, który realizowałem różni się od obecnych produktów tym, że działa po stronie klienta. To znaczy, że zakładam pewne niedoskonałości skanerów antyspamowych oferowanych przez dostawców usług internetowych, w wyniku których mały procent spamu zostaje dostarczony do naszych lokalnych skrzynek pocztowych.

Spamkilla – jest rozszerzeniem dla programu Thunderbird, które analizuje wszystkie nowe wiadomości trafiające do skrzynki pocztowej, jeżeli zawierają one załączniki o określonej wielkości w postaci plików graficznych taka wiadomość jest poddawana dalszej analizie.

Kolejnym etapem procesu jest przesłanie plików graficznych wyłuskanych z wiadomości na serwer, który dokonuj ich analizy i zwraca pewne prawdopodobieństwo z jakim dany plik jest spamem. W tym momencie spamkilla na podstawie tej informacji i wcześniej zdefiniowanego progu wykonuje odpowiednią akcję. Jeżeli zwrócone przez serwer prawdopodobieństwo jest większe od zdefiniowanego przez użytkownika wiadomość trafia do specjalnego folderu, w przeciwnym wypadku nie są podejmowane żadne akcje.

Rozwiązanie z wykorzystaniem architektury klient-serwer jest bardzo korzystne, ponieważ pozwala na gromadzenie statystyk i dodatkowych informacji o spamie, umożliwia na dynamiczne zmiany algorytmów w sposób nie widoczny dla końcowych użytkowników, a dzięki dużej mocy obliczeniowej można tworzyć znacznie bardziej wyrafinowane metody wykrywania spamu. Co więcej uwalniamy użytkownika od konieczności aktualizacji bazy danych zawierającej odpowiednie sygnatury spamu. Poniższy rysunek przedstawia schemat działania:



Technologie

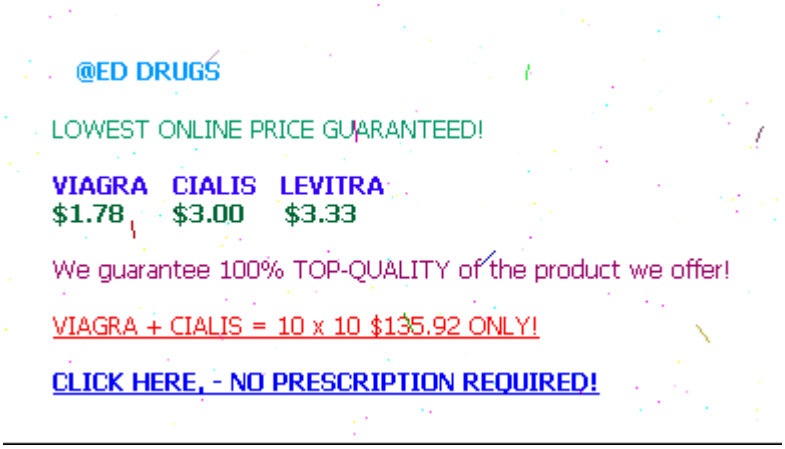
W projekcie wykorzystałem następujące technologie:

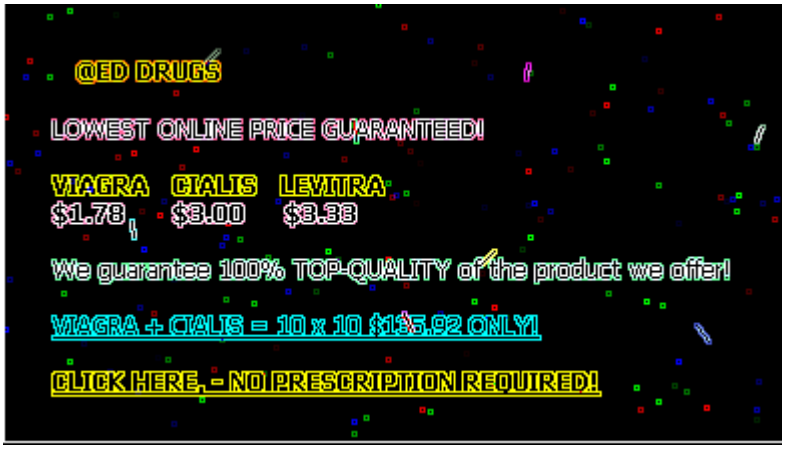
1. Po stronie serwerowej wykorzystałem technologię servletów Javy, wybór ten był warunkowany istnieniem dużej ilości bibliotek ułatwiających manipulowanie grafiką. W szczególności używałem otwartej biblioteki JIU - Java Imaging Utilities (http://schmidt.devlib.org/jiu/). Kolejnym istotnym elementem projektu jest silnik OCR pozwalający na przetwarzanie plików graficznych do formatu tekstowego. Mój wybór padł na otwarte i darmowe oprogramowanie o nazwie Gocr (http://jocr.sourceforge.net). Z testów które przeprowadziłem wynika, że Gocr najlepiej radził sobie z oryginalnym nie poddanym żadnym obróbkom spamem. Co więcej prędkość jego działania była bardzo zadowalająca.
2. Po stornie klienckiej zostało stworzone rozszerzenie dla programu pocztowego Thunderbird. Część graficzna została zbudowana w języku XUL (dialekt XMLa opracowany specjalnie na potrzeby tworzenia elementów GUI). Natomiast logika aplikacji powstała w języku JavaScript (produkty fundacji Mozilla udostępniają szereg dodatkowych interfejsów pozwalających na wykonywanie na przykład operacji plikowych itp.)

Opis problemu

Głównym celem moje projektu było polepszenie parametrów plików graficznych, w taki sposób, aby po przetworzeniu ich przez oprogramowanie OCR uzyskać dokładniejszą informację o zawartym tekście. Jest to kluczowa punkt, ponieważ od wyniku uzyskanego w tej fazie zależało całe dalsze przetwarzanie.

Jak zostało napisane wcześniej spamerzy stosują różne techniki, których celem jest oszukanie oprogramowania służącego do rozpoznawania tekstu. Na szczęście większość z tych technik często się powtarza i jest w miarę prosta do wykrycia. Ulubioną metodą spamerów jest wprowadzanie celowych zakłóceń do grafiki. W większości przypadków są to pojedyncze piksele, lub grupy kilkunastu pikseli rozmieszczonych w losowych miejscach w pliku. Dla człowieka takie zakłócenia są niezauważalne i nie powodują problemów w interpretacji informacji, natomiast oprogramowanie OCR może mieć problemy z interpretacją takich krzaczków. Poniżej zostały przedstawione przykłady (oryginalny plik, oraz po procedurze wykrywania zakłoceń)





Oczywiście istnieją też inne rodzaje zakłóceń np. stosowanie bardzo jasnych kolorów dla czcionek, czy różnokolorowe poprzeczne paski. Z tego powodu nie da się stworzyć uniwersalnego mechanizmu wykrywania spamu. Co jest jednak ciekawe z niektórymi typami zakłóceń oprogramowanie OCR radzi sobie bez problemu.

W dalszej części sprawozdania skupie się na technikach poprawiających jakość obrazków ze szczególnym uwzględnieniem przypadków przedstawionych wcześniej.

Ogólna idea jest taka, aby wykryć wszystkie możliwe obszary na obrazie, następnie je przeanalizować, a później usunąć z oryginalnego pliku te, które zostały uznane jako zakłócenia.

W początkowej fazie tworzenia oprogramowania do wykrywania obszarów wykorzystałem technikę polegającą w pierwszej kolejności na znajdowaniu krawędzi (z użyciem operatora Convolution i odpowiednim jądrem), a następnie łączenia tych krawędzi i tworzenia obszarów. Jednak w późniejszym okresie zastosowałem znacznie prostszą metodę, która daje takie same wyniki. Otóż wystarczy przejść z trybu RGB to trybu szarości, a następnie na podstawie histogramu dobrać odpowiednio parametry i dokonać dyskretyzacji obrazu. Po takiej operacji wszystkie widoczne wcześniej piksele mają kolor czarny, natomiast pozostałe obszary są koloru białego.

Kolejnym etapem jest analiza uzyskanych wcześniej obszarów, warto zaznaczyć, że nie wszystkie obszary są jednakowo interesujące. Z punktu widzenia mojego algorytmu najbardziej interesujące są te o liczności nie przekraczającej 30 pikseli (gdyż zazwyczaj są to zakłócenia, znaki interpunkcyjne lub wypunktowania i inne elementu utrudniające proces rozpoznawania tekstu). Dla takich danych uruchomiona zostaje procedura znajdowania sąsiadów. Jeżeli okaże się w sąsiedztwie o zdefiniowanym wcześniej promieniu nie występują żadne inne obszary to usuwam takie elementu z oryginalnego obrazu. Najtrudniejsze do znalezienia są zakłócenia wprowadzane bardzo blisko innych poprawnych znaków, ponieważ tak na prawdę na tym etapie algorytm nie ma wiedzy pozwalającej na stwierdzenie czy jest to znak i zakłócenie, czy poprawny znak taki jak na przykład „Śćżźą” itp. Istnieje też pewna grupa połączeń znaków sprawiająca szczególnie duże problemy na przykład „L 0” czy „1 0” itp. ponieważ najbliższa odległość między tymi znakami jest często większa od tej zdefiniowanej w algorytmie (5 pikseli). W związku z tym procedura rozpoznawania tekstu jest wywoływana dwukrotnie. Najpierw na oryginalnym obrazie, a następnie na obrazie, który przeszedł przez algorytm poprawy jego jakości. Po otrzymaniu wyniku z silnika OCR trafia on do sekcji obliczającej prawdopodobieństwo tego, że na obrazie znajdują się niechciane treści. W tym celu otrzymane wyrazy są porównywane z wyrazami znajdującymi się na czarnej liście. Jednak nie jest to takie zwykłe porównanie, ponieważ wiadomo, że pozyskane teksty mogą być częściowo niekompletne, lub zawierać błędy. W tym celu jest wykorzystywany algorytm Levenshteina, który zwraca odległość pomiędzy dwoma wyrazami (mierzoną w liczbie przestawień, jakie należy wykonać, aby oba łańcuchy znakowe były takie same), która przekłada się na wartość prawdopodobieństwa. W końcowym etapie algorytm zwraca większą z wartości prawdopodobieństwa uzyskaną po przetworzeniu oryginalnego pliku i jego ulepszonej wersji. Następnie wynik wraca do klienta, a ten dokonuje interpretacji.

Wyniki

Okazało się, że dzięki zastosowaniu prostych przekształceń opisanych wcześniej można znacząco poprawić jakość obrazu wejściowego, co przekłada się na ilość i jakość informacji uzyskanej w procesie rozpoznawania tekstu przez oprogramowanie typu OCR. Niekiedy wystarczyło aby silnik OCR wykrył 2 lub 3 dodatkowe litery, aby algorytm z dużą pewnością mógł zaklasyfikować wiadomość jako spam.

Poniżej przedstawiam kilka wyników(są to logi z rzeczywistych sesji)

Oryginalny obraz

_ ; , ' , '',' '

_ed drugs r

lowest onl, ine price gmranteed! , r

via?r cialrs levnra

$_78l _$_o,o $_33 , _

we quaraniee looo_o to_-qualny o_ihe produci we o_r!

***_x _ ***

_' , ,

Obraz po przetworzeniu

: _ed drugs

owen online price cmrameed!

viagra cialis levitra

$1_8 $3.o,o $3.33

we quaraniee looo_o top-qualny o_ihe produci we o_r!

_x

Oryginalny obraz:

_ _\ ' _' ' k ' ' ' ' ' _ , ',' , '

, ' '

, legal rx medications ' ,

'c ' _,lih_'_c7h_'_ 'c !_ih_'_lii' 'c le'_,l 7h_'_

v propcia v maman floma

'c zolon ,'c _vellbutrin sr , _ _ ' _,

on _sale n'ow! , ,

lo_ven prices online cuaramged!

' _ , '

_, ,

Obraz po przetworzeniu

legal rx medications

'c ' _,lih_'_c7h_'_ 'c !_ih_'_lii' 'c le'_,l 7h_'_

^ propecia ^ maxman ^ flomax

_'c zolon _'c _vellbutrin sr

on _sale now!

lo_ven prices online cuaramged!

Problemy

Większość problemów, które napotkałem pisząc projekt byłą związana z częścią kliencką. Dużą utrudnieniem była konieczność wykorzystywania języka JavaScript, który jest językiem skryptowym, co znacznie utrudniało proces pisania i znajdowania błędów. Poważny utrudnieniem był także brak dobrej dokumentacji dotyczącej interfejsów i obiektów oferowanych przez Thunderbirda(problemy opisane w poprzednich sprawozdaniach).