Categorizzazione degli host in ntopng

Matteo Loporchio, Marco Pacini, Lorenzo Vangi

A.A. 2014/2015

Sommario

In questo documento sono descritte le modifiche al codice sorgente di **ntopng** al fine di estendere quest'ultimo con un sistema di categorizzazione degli host mediante il servizio **Google Safe Browsing**. Il risultato delle modifiche apportate al codice sorgente originale è visibile su **GitHub** all'indirizzo `https://github.com/ntop/ntopng/commit/81642359ad927d66635872dc81c69cf98ff3cef6` ed è stato definitivamente approvato come parte integrante di **ntopng**.

1 Introduzione

Il servizio **Google Safe Browsing** offre la possibilità agli utenti di ottenere informazioni circa l'affidabilità dei siti web, avvertendoli ogni volta che si cerca di visitare una pagina web che può contenere malware od essere usata per tentativi di phishing. È possibile interfacciarsi con il servizio tramite un'API pubblica che prevede l'utilizzo di una chiave alfanumerica ed univoca, ottenibile tramite registrazione a partire dall'indirizzo `https://developers.google.com/safe-browsing`.
Il compito degli autori del presente documento è stato quello di estendere il codice sorgente di **ntopng** in modo tale da permettere la classificazione degli host in due categorie, ovvero *reliable* e *malware*, a seconda della risposta fornita dalla API.
Poichè la chiave per l'API è univoca e strettamente personale, nella versione più recente di **ntopng** è stata implementata la possibilità di utilizzare il servizio di categorizzazione lanciandolo con il comando `ntopng -c <categorization_key>`, inserendo al posto di `<categorization_key>` la propria chiave.

2 Modifiche preliminari

Per abilitare la funzionalità di categorizzazione all'interno di **ntopng** si è reso necessario modificare alcune righe del file sorgente `Prefs.cpp` contenente le impostazioni di base del programma. In particolare, il costruttore della classe `Prefs` è stato modificato in modo che la variabile `categorization_enabled` venga inizializzata a `true` e che la variabile `categorization_key` venga inizializzata con la chiave dell'API di Google Safe Browsing. È possibile definire tale chiave come macro all'inizio del file stesso, tramite la riga `#define DEFAULT_CATEGORIZATION_KEY "contenuto della chiave"`.

3 Effettuare la richiesta all'API

Per ottenere informazioni circa l'affidabilità di un dato host, è necessario inviare una richiesta all'API di Safe Browsing. L'interazione avviene mediante il protocollo HTTP (nella versione v3) e per ottenere una

risposta è sufficiente invocare il metodo **GET** a partire da un URL specifico. In particolare, il formato dell'URL è il seguente: `https://sb-ssl.google.com/safebrowsing/api/lookup?client=CLIENT&key=APIKEY&appver=APPVER&pver=PVER&url=URL`.
Nella stringa precedente, `CLIENT` identifica il nome dell'applicazione che sta eseguendo la richiesta, il parametro `APIKEY` è ovviamente la chiave univoca, i parametri `APPVER` e `PVER` sono rispettivamente le versioni della propria applicazione e del servizio utilizzato ed infine il parametro `URL` corrisponde all'URL che vogliamo verificare, opportunamente codificato (si veda a tale proposito la sezione 3.3).

3.1 Implementazione del metodo GET

La funzione che realizza la richiesta GET HTTP è stata definita nel file sorgente `Utils.cpp` nel seguente modo:

```
char* curlHTTPGet(char *url, long *http_code)
```

Due sono i parametri della funzione: il primo, `url`, è la stringa corrispondente all'indirizzo su cui eseguire la richiesta, mentre il secondo, `http_code`, è un intero che viene passato per riferimento e nel quale la funzione memorizza il codice di risposta HTTP una volta eseguita l'operazione. Si noti che per realizzare tale funzionalità gli autori si sono serviti della libreria **libcurl**.
Nel nostro caso, ad ogni invocazione di questa funzione, il parametro `url` avrà sempre la forma descritta sopra, dato che la funzione viene chiamata esclusivamente per ottenere una risposta dall'API di Google Safe Browsing.

3.2 Funzioni di supporto per `curlHTTPGet`

A questo punto, è doveroso sottolineare il fatto che, come comportamento di default, la libreria **libcurl** prevede la stampa sullo *standard output* del risultato delle proprie operazioni. Tuttavia, per ragioni pratiche, gli autori hanno ritenuto opportuno ridirezionare l'output all'interno di un'apposita stringa, restituita come risultato dalla funzione `curlHTTPGet`.
A tale scopo è stata definita, sempre all'interno del file `Utils.cpp`, un'ulteriore funzione di supporto denominata `writeFunc`:

```
static size_t writeFunc(void *ptr, size_t size, size_t nmemb, String *str)
```

In questa funzione i parametri sono conformi alla forma richiesta dalla libreria **libcurl** e, più nello specifico, al metodo `curl_easy_setopt`, che permette di definire vari aspetti del comportamento della libreria, tra cui anche la redirezione dell'output. Per la definizione di `writeFunc`, gli autori hanno fatto ricorso alla definizione di una *struct* denominata `String` che contiene un `char *` rappresentante la stringa vera e propria, sia un intero che rappresenta la lunghezza della stringa. Inoltre, all'interno del medesimo file, è stato definito un metodo (statico) di supporto:

```
static void newString(String *str)
```

che inizializza una nuova `String` ponendo il `char *` uguale a `NULL` e la sua lunghezza pari a 0.

3.3 Ulteriori funzioni di supporto

Affinché le richieste all'API di Safe Browsing abbiano esito positivo, è necessario che l'URL dell'host da classificare sia correttamente formattato: esso deve essere codificato in maniera tale che i caratteri speciali in esso contenuti, come ad esempio `:` o `/` siano rimpiazzati con il rispettivo codice ASCII preceduto dal simbolo `%`. Per questo motivo gli autori hanno deciso di implementare all'interno del file `Utils.cpp` una serie di funzioni utili per la codifica degli URL.
In particolare, il metodo principale è il seguente:

```
char* Utils::urlEncode(char *url)
```

Come si può intuire dal nome, `urlEncode` si occupa di trasformare il parametro `url` in un URL codificato. Per fare ciò, esso si avvale a sua volta di due metodi aggiuntivi, denominati `to_hex` e `alphanum`: il primo dei due, preso in ingresso un carattere `c`, lo converte in formato esadecimale; il secondo, invece, verifica se è effettivamente un carattere alfanumerico (vale a dire una cifra fra `0` e `9` o una lettera fra `A` e `Z`).

4 La classificazione

Prima di invocare la funzione `curlHTTPGet` è necessario preparare l'URL da passarle come parametro. Ciò è possibile tramite una semplice concatenazione di stringhe. Più dettagliatamente, questo compito viene svolto nel metodo:

```
void Categorization::categorizeHostName(char *_url, char *buf, u_int buf_len)
```

dichiarato nel file `Categorization.cpp`. Si tratta del metodo principale del nostro sistema di categorizzazione. Riportiamo di seguito una breve descrizione dei parametri e delle operazioni che esso effettua, nell'ordine in cui esse vengono svolte.

1. La funzione viene invocata passando come parametri: una stringa `_url`, corrispondente all'URL (non ancora codificato) da classificare (ad esempio: `http://www.google.com` oppure `http://adv-inc-net.com`), una seconda stringa denominata `buf`, nella quale verrà salvato il risultato della categorizzazione, e un intero `buf_len` che rappresenta la dimensione di `buf`.

2. Vengono raccolte tutte le informazioni necessarie alla costruzione dell'URL da passare a `curlHTTPGet`, ovvero: `CLIENT`, `APIKEY`, `APPVER`, `PVER` e `URL` (si veda a tal proposito la sezione 3). Si noti che `URL` viene prodotto invocando la `Utils::urlEncode` passandole come parametro `_url`.

3. Una volta costruito l'URL, esso viene passato come parametro a `curlHTTPGet`, che salva il codice di risposta in una variabile denominata `replyCode` (il cui valore è inizialmente 0) e memorizza la stringa di uscita in un apposito buffer.

4. In base al codice di risposta ottenuto, si procede con la classificazione vera e propria. Se il codice di risposta è pari a `200 OK` significa che il nostro sito è stato classificato come pericoloso e dunque otterremo la stringa `malware`. Altrimenti, se il codice è `204 NO CONTENT`, significa che il sito in questione è affidabile. Tuttavia la richiesta all'API produrrà una stringa vuota, il che non ci aiuta nel caso in cui volessimo mostrare a video i risultati. Gli autori hanno deciso pertanto di sovrascrivere tale stringa vuota e di stampare comunque la stringa `reliable`, decisamente più significativa. Se il codice di uscita è invece `400 BAD REQUEST`, significa che l'URL su cui abbiamo eseguito la GET è

stato formattato in maniera sbagliata, mentre un altro numero diverso dai precedenti è caratteristico di un'operazione non andata a buon fine. Segnaliamo infine il caso particolare in cui il codice continua a valere `0` anche dopo l'invocazione del metodo `curlHTTPGet`: in situazioni come questa il malfunzionamento è imputabile solamente al fallimento della libreria **libcurl**.

5. Il contenuto del buffer di uscita viene copiato nel parametro `buf` di `categorizeHostName` stessa. Inoltre, vengono stampati a video dei messaggi contenenti l'esito dell'operazione e l'URL su cui è stata effettuata la richiesta. Come ultima, ma non meno importante, operazione, il risultato viene salvato dentro il server **Redis** sotto forma di chiave.