Spell Checking or Key Logging?

Software Analysis on Grammarly the Web Browser Plugin for Grammatical Correction

Jianjie Liu

1st December 2017

1 Abstract

Spell checking software has becoming more promising as language processing techniques advance. These software gain a large pool of users from the web browser plugin platform, enabling users to spell check the content when writing emails, posts and comments on social media. On the surface, these software are checking grammar. Behind the cover, they are monitoring what the users are typing and send the data back to a server for correcting the grammatical errors. These browser plugins are alternative form of key loggers, software that record each key a user presses on the keyboard.

The controversy of using such software is that the users are exposing their privacy in exchange for the service. At worst, the spell checker may be accidentally recording sensitive information like username and password when logging into a website or credit card information when purchasing online.

This paper will be a software analysis of one of the popular spell checking web browser plugin, Grammarly. We will be performing both a static analysis and reverse engineering of the minified source code. We will also perform a dynamic analysis through capturing network packets this software may send out from the client. The goal in these analyses is to find out what information Grammarly is collecting.

2 Introduction

Spell checking, although it is questionable on how effective it is in improving user's writing, has become an essential tool on the web. Popular websites and web browsers have support for spell checking users' input. However, these native supports are deemed too simplistic and primitive to resolve complicated issues in improving writing style and word choices. Many users turn to the more sophisticated grammar-checking software, such as Grammarly, to enhances their experience surfing the web.

Grammarly is one of the most popular web browser extensions that check the grammatical errors when users are writing emails, comments or posts on social media through the web browser. The general software architecture for this type of software is a client-server model. The language-processing algorithm that powers the writing enhancement features resides in an external server. The plugin that users (from here and on, user and client are used interchangeably) install onto their web browser is an interface to intercept user writing and send them to the algorithm server for grammatical correction.

A keystroke logging software captures the key struck on a keyboard. A key-logger can be found in malwares to secretly record any sensitive information users may input into the infected device. Such key monitoring software can become the worst malware to encounter as it can bypass any security measures to obtain users' sensitive information. It can record in raw text what website a user types into the browser and the login credentials used authenticate into the site.

In many aspects, a spell checker can also be classified as a key-logger. Both monitor user inputs and send the captured information to elsewhere for further processing. Despite that the goal was to analyze syntax errors, spell checkers like Grammarly can accidentally log sensitive information. Therefore, we need evaluate the following aspects in our software analysis.

1. Information Collection

   We need to find out if the application is capturing everything users input when using the browser, or it will filter out sensitive information. We can analyze if the software selectively logs user input on specific HTML elements of a visited site.

2. Connection Protocols

   The security of users' information relies heavily on the protocol that the application uses to transfer data across domains. We can find out if the application is using any third party frameworks and libraries to establish connection between the client and the server. Then we evaluate the risk of using these frameworks. It is also to our interest to see if secured protocols like TSL is used to transport data within the traffic.

3 Message to Community

Today, the Internet is a portal of convenience that allows us to find the quickest and easiest way to perform our errands. We can download software or use online services to do the bidding for us, such as using spelling checkers on a web browser. However, this convenience comes with a price. Users are too reckless in use of software that they are willing to trust a service without evaluating the possible risk of leaking their personal information to such service. Companies are unwilling to inform users the different information they are collecting from their clients, and perhaps only hinting the price of usage in their terms of service, an agreement largely neglected by most users.

The following is excerpted from the Grammarly's privacy policy.

> The Site and the Software also collect and receive information from your computer or mobile device, including the activities you perform within your account, the type of hardware and software you are using (for example, your operating system, word processing or other productivity software and browser type) and information obtained from cookies. When you access the Software, that application will request access to certain information on your computing device.

The language here is especially obscure when this agreement discusses what information the "Software" (i.e. the web browser plugin) is collecting when using it. It did not explain what this "certain information on your [user's] computing device" is or discuss the possible privacy violation relating to key logging.

Currently, more than ten million users have installed Grammarly on their Google Chrome browser. Most users are not aware of the key-logging feature that grammar checkers rely on to implement their service. It is even difficult for them to evaluate the danger of information leakage through using this type of software.

4 Static Analysis and Reverse Engineering

The source code is obtained from installing Grammarly's extension for Google Chrome browser. However, the code is compiled into minified JavaScript file, where the originally separated source files are put together into one file. Functions, objects and input variables in this file are renamed and reduced to a single alphabet (minified source is found in the "src" folder under support materials). For example "someFunction(input1, input2)" will become "a(b,c)". We need to apply a bit of reverse engineering to fully understand the source code. Our goal for this analysis is to understand how Grammarly collects user inputs and where the software is sending user data. To do so, we need to isolate and analyze two primary modules: the one that monitors user's keystrokes and the other that sends user input to a external server for grammatical corrections.

In addition, the minified source code is further analyzed using Veracode to scan for possible common vulnerabilities within the coding structure (e.g. cross-site scripting). The result shows that there are no significant findings in common software and JavaScript vulnerabilities. This report is located under the supporting materials.

4.1 Procedure

The minified source code has numerous files configuring the user interface. We are only interested in the implementation of the background service of the application, namely the file "Grammarly.js". Although the minified format renamed most of the variable and function names, fortunately some of key class methods are untouched. We can begin by opening up the source file in a text editor and search for keywords in the file pertaining to keystroke monitoring and server connection. These keywords can be "listener", "connect", "input", "socket", "http", etc.

After identifying the particular functions or variable name of interested action, such as "inputListener()", we will search up the code stack for the parent structure that contains the keywords or other functions of relating functionality, such as "createInputListener()" until we find a root JavaScipt object that contains most of functions of interest we have identified before. We then extract this root object from the original source file into a separate for deeper analysis. We can trace the call stack during the object's initialization, record the names of any third party library used, and since it is written in JavaScript, we can analyze the individual function to figure out what it is doing. Keep in mind that this is not a full reverse engineering analysis. We are only interested in two features of the software: how the grammar checkers log user's keystrokes and where does it send the collected data.

4.2 Findings

There are three key modules isolated from the minified source code: "inputListener", "webSocketApi" and "telemetry". These are objects representing individual task that composes the general usage of the software. For example, to spell check user's input, the software relies on "inputListener" module to monitor user's keystrokes when he or she is writing an email. Then it uses the "webSocketApi" to send captured text to a server for grammatical analysis.

Often, these modules have methods used by other parts of the software. We will refer these parts of software as "clients" of the module (from the module's perspective, they are "users" of the module's implementation). It is essential that we investigate how clients are using each module. For instance, does Grammarly use "inputListener" to track keystrokes whenever users type through the web browser or only when they enter through specific input boxes in a particular site?

4.2.a InputListener Module

This module contains methods for logging user inputs and classifying the associated key code. It also has the starter and the stopper method to control when to listen for input. These methods rely on the event emitter and listener structure (see Reference Link #5 for details of emitter in Node.js). This module contains an emitter, and when the methods in this module detect an event happening on the browser (e.g. user presses the enter key to submit a form), the corresponding method will send an event along with some data (this can be the pressed keystroke code) to the emitter. The client of this module will have a copy of this emitter and will be listening for any event the emitter transmits as well as the data that comes with the event.

"grammarlyGhost.js" is a client of the "inputListener" module. This client creates an invisible HTML element (thus "ghost" element) wrapping around specific HTML elements of a web site, usual textboxes or input boxes. Then the client will initiate an instance of inputListener module as "dom_1". When user clicks into the textbox that contains this ghost HTML, the client will start key logging user's input. It will stop when user leaves the textbox by clicking outside the textbox or submitting whatever they typed into the textbox.

4.2.b WebSocket Module

This module is responsible for sending string to Grammarly's server for textual analysis, including grammatical correction, word choice enhancement and plagiarism detection. The connection to server is established using WebSocket library (see Reference Link #6). Presumably, the server side is also implemented using the same library. Most of the methods in the module take in "changedText" as parameters, indicating there is an intermediate step for preprocessing the raw text from monitoring users' keystrokes. This step is implemented in "processInput.js".

The webSocket connection is established in "connectWebSocket.js" where the server domain is stored in a constant named "options.url". Unfortunately, we did not find either a domain name or IP of webSocket server in the source code. It is likely that there are multiple instances of webSocket server for traffic dispersion, and the software dynamically connects to one of the instances with the lightest traffic load. Thus there is no need to hard code the webSocket server IP when it is dynamically supplied at the runtime of the software.

The webSocket connection does not rely on HTTP or HTTPS protocol. We need to capture network packets at runtime of the software to conclude if data sent through this

connection is encrypted. We will discuss the security of data transport in the following dynamic analysis.

4.2.c Telemetry Module

This module is another input for collecting user information. The telemetry module records the software's runtime issues and tracks users' activities on the software's user interface. These messages are transmitted to the domain "f-log-extension.grammarly.io" through HTTPS protocol. The "log1.LogLevel" constant used in the methods of this module also classifies the message into three categories: "warning", "error" and "info".

We can suspect that the use of this module is to remotely collect information on software crashes, dependency errors, connection issues and trigger server side events in response to user activity. The information collected by the telemetry module is perhaps purely functional data that helps to analyze software performance and log errors in the software. This information should not pose a threat to user privacy.

5 Dynamic Analysis

The goal of this analysis to provide proof of concept to the functionality of the three modules we reverse engineered in the above section. We will analyze the HTML injection that Grammarly performs onto its supporting domains and find evidence of key logging from capturing network packets.

5.1 Procedure

This dynamic analysis is performed in a virtual machine. We will be installing the Grammarly browser plugin on a Firefox browser. We will need to setup mitmproxy to monitor HTTPS traffic to Grammarly domains by installing the mitmproxy certificate onto browser and route the traffic in the browser through the proxy. Lastly, we also have to use wireshark, a network sniffing software, to capture network packets on the virtual device's Ethernet interface.

The analysis begins with browsing the domains supported by Grammarly, including Gmail, YouTube, and Tumblr (collectively, we will refer to these as supported domains). We will be writing on these sites and exam the outgoing packets through both the proxy and the network sniffer for destination domain and their contents. Then we will analyze the software on non-supporting sites in the similar manner. We can do this by creating a simple HTML page with an input box and open this local HTML file on browser.

5.2 Findings

As a trivial proof of concept, we confirmed that the Grammarly's web plugin cannot run offline when its service stops after internet access was disconnected. On supporting domains, we discovered Grammarly's "ghost" HTML tag inside specific textboxes where user will be writing emails, comments or blogs. When the user began typing in these textboxes, Grammarly was logging their input. We could attest to this assumption by examining the network packets. A large amount of traffic was sent to the IP "34.194.72.X", which belongs to Amazon Web Service. Presumably, this IP is where Grammarly host its WebSocket server for receiving client input for grammatical analysis. These packets are encrypted using TSL protocol, so we could not fully confirm that they contain user keystrokes, but it was highly likely considering how closely the time between these packets were sent and user was typing.

Examining the HTTPS traffic on the proxy, we can also discover POST requests to the domain "f-log-extension.grammarly.io". The request body contains mostly event message that was triggered when user clicked on the interface that Grammarly injected into the supporting sites. Like we discussed before, these messages do not relate to any private information and are likely used for debugging purposes.

For non-supporting sites, there was no evidence of Grammarly injecting "ghost" HTML tag in any of the input HTML elements and key logging user inputs. However, Grammarly did record the domain of the unsupported sites and sent to the "f-log-extension.grammarly.io" domain.

6 Conclusion

From result the analysis above, we can conclude that Grammarly monitors user input strictly on the specific textboxes in supporting sites (i.e. pertaining to those containing the "ghost" HTML tag). The software also logs crash information and activity event when user triggers them. All transmitted data to Grammarly domain is encrypted using TSL protocol.

The implication of the finding is that Grammarly does not track sensitive information including user authentication and financial information. The software makes it clear when it is monitoring user input by showing its icon inside the textbox in which the user is typing. Unless user is intentionally transmitting their private information through a spell checked textbox, for example, sending passwords through Facebook chat or Gmail, Grammarly will not receive them.

Nevertheless, any message user input into these textboxes with spell checking, Grammarly will monitor it. It will be a different debate to discuss whether to classify the content of these messages as sensitive information and determine if Grammarly has the right to store them. It will be a different investigation to understand if Grammarly has kept a bank of all of users' messages it spell checked. However, Grammarly does protect user data with industry standard encryption and does not record sensitive information that user enters into the sites the visit.

Users should not take free services or application for granted. They are not truly free, and often they record user's information in exchange for the service they provide. In most cases, the moment a user accepts a software service, the user agrees to its terms of service, which often declares that users are abandoning the legal right to own any information the software collects from the user himself.

In short, it should be made clear to users of Grammarly that it is spell checking what they writing on the Internet and at the same time monitoring or perhaps even recording them. Users should be advised to uninstall the software if they feel their privacy is threatened.

7 References

1. https://www.grammarly.com/privacy-policy
2. https://www.grammarly.com/terms
3. http://www.ethicalhackersclub.in/2016/09/grammarly-user-this-is-for-you-now.html
4. https://gist.github.com/subudeepak/9897212
5. https://nodejs.org/api/events.html
6. https://github.com/websockets/ws
7. http://searchsecurity.techtarget.com/definition/keylogger
8. https://wiremask.eu/articles/xss-keylogger-tutorial/