Development of Privacy-Preserving Online Monitoring Framework for Online healthcare Applications

Minsoo Kim and Youna Jung
Department of Computer and Information Sciences
Virginia Military Institute
Lexington, Virginia, United States
e-mail: kimm@vmi.edu, jungy@vmi.edu

Abstract— **Privacy concern is one of the biggest obstacles in widespread adoption of online monitoring services on e-health applications, even though online monitoring can significantly improve the accuracy and quality of e-health applications. To prevent privacy loss during online monitoring, we have preliminarily proposed a privacy-preserving online monitoring framework (PPoM) that enables healthcare providers and patients to intuitively specify their own privacy policies and enforces patients' privacy policies in systematic manner during monitoring. For practical use of the PPoM framework, in this paper, we present the prototype development of the PPoM framework in detail. For the better understanding, we provide example usages from the standpoints of both healthcare providers and patients. To prove the performance of the developed prototype, we present evaluation results.**

Keywords- Privacy protection; online monitoring; framework

I. INTRODUCTION

Monitoring is one of the essential techniques to evaluate and enhance the performance of e-health applications by tracking and analyzing the online activities of patients, such as mouse clicks, frequency of use of an application, time spent in a particular page, media viewed, page navigation sequences, content entered into a textbox, location information, whether a mobile device is being used, etc.

Due to the sensitivity of the information that e-health applications often deal with, however, the protection of user privacy is critical. Control over the sharing of this information is of the utmost importance and urgency because indiscriminate monitoring, if inconsiderate of user privacy, may result in private health data being used for unwanted purposes and/or shared with unknown people [1][2][3]. It is therefore urgent and critical to enable the monitoring identifiable user data while protecting user privacy.

To this end, we have preliminarily proposed the PPoM framework [4] and the Health Insurance Portability and Accountability Act (HIPAA) Compliant Privacy Policy Language for e-health Applications [5]. The PPoM framework enables healthcare providers to collect necessary information without violation of patient's privacy preferences and HIPAA regulations and allows patients to enforce their own privacy preferences on the user side. To realize the proposed idea, we developed a prototype of the PPoM framework. The ultimate goals of the prototype development are as follows:

- For non-IT administrators of e-health applications
  - provide an intuitive way to describe privacy policies for their applications
  - provide an easy way to monitor patients' activities on their applications and collect patients' data without serious privacy breach
- For patients
  - provide a way to systematically verify an application's compliance with mutually agreed policies and HIPAA
  - Provide a way to rigorously protect their private data based on their preference on the user side

The rest of this paper is organized as follows. In Section II, our preliminary work is briefly introduced. In Section III, we present the architectural design of the PPoM framework and describe details about prototype development with example usages. In Section IV, we explain our test environment and present evaluation results. Our conclusion and future work are described in Section V.

II. PRELIMINARY WORK

To address the privacy issues on e-heath applications conducting online monitoring, the PPoM framework has been proposed [4]. In this section, we briefly introduce the overall architecture and the operational flow of the PPoM framework. As shown in Fig. 1, the PPoM framework consists of four components: the PPoM Service, the PPoM Browser, the PPoM Tools (PPoMT), and the HIPAA-Compliant Privacy Policy Language [5].

- *PPoM Service* – It is an online monitoring service that gathers only authorized user/usage data that users allow to monitor. By specifying user policies, patients can determine which data can be monitored. Then, the PPoM Service selectively collects user/usage data based on user policies. Unlike the existing monitoring services where user data are collected based on an application's policies and the policies are enforced by the application itself, the PPoM Service provides a way to refer user policies during online monitoring in a systematic manner rather than simply providing a written agreement.

 ISBN: 978-1-61208-511-1

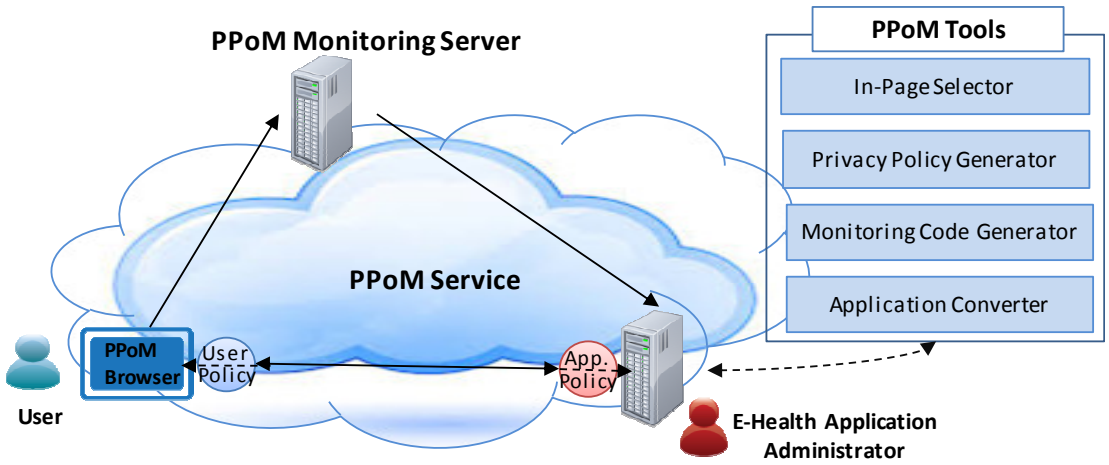

Figure 1. Overall Architecture of PPoM Framework.

- *PPoM Browser* – The privacy of patients must be protected even if a user is exposed to untrustworthy e-health applications that conduct indiscriminate monitoring in violation of user policies. Towards this end, the PPoM Browser understands a patient's privacy preferences, presents all user data being monitored, and protects a patient's privacy on the user side by blocking outgoing messages which contain data he/she does not want to disclose.
- *PPoMT* – Although patient monitoring is critical in e-health applications, it is difficult for healthcare providers to develop monitoring-enabled applications with application policies due to lack of professional IT knowledge. The PPoMT helps non-IT health professionals by enabling them to specify privacy policies for their applications, and to convert existing applications into privacy-preserving applications.
- *HIPAA-Compliant Privacy Policy Language* – Existing general-purpose privacy policy languages, including P3P [6], APPEL [7], and XPref [8], focus on generic user/usage data to be used for a variety of online applications and do not give careful consideration to health data. It is therefore impossible for both patients and e-health providers to precisely specify their privacy policies on health data in fine-grained level, and in turn, it lowers the performance of the PPoM framework. To address the lack of consideration of health data in existing privacy policy languages, the PPoM framework uses the HIPAA-compliant Privacy Policy Language employing the HIPAA profile [5], which allow an e-health provider to specify a privacy policy related to HIPAA regulations and enable a patient to specify his/her preference on health data in detail.

To use the PPoM framework, an online healthcare provider first needs to upload the source code or enter the URL(s) of his/her e-health application to the PPoMT and then select objects to be monitored and the corresponding privacy policies through the user-friendly interfaces generated by the In-page Selector. The Privacy Policy Generator then creates the application's policies by analyzing selected monitoring data and policies, while the Application Converter produces updated source code by inserting monitoring code generated by the Monitoring Code Generator into the original source code. The application policies and the updated source code must be deployed in the application server.

A patient now can use e-health applications without privacy concern. Whenever a patient enters a URL of e-health applications, his/her PPoM browser compares user policies and application policies. If they match, the application server sends PPoM-enabled pages which privacy-aware monitoring code is embed in. As the patient interacts with the application, the PPoM browser displays all user/usage data being monitored so that the patient can verify privacy protection during online monitoring. The monitoring code inserted in webpages collects only authorized user data based on user policies. The PPoM browser will block outgoing messages that violate the patient's policies.

III. DEVELOPMENT OF THE PPOM FRAMEWORK

To realize the proposed idea for privacy protection during online Monitoring, we developed a prototype of the PPoM framework. In this section, we describe details of implementation of each component. Note that the prototype of the PPoM framework is developed using mainly PHP and JavaScript. PHP is used for developing the backend of the PPoM Browser and the PPoM Tools while JavaScript including jQuery functions, HTML5, and CSS3 are mainly used for developing the user interfaces (UIs) on the client side. We use MySQL as a database of the PPoM service.

A. PPoM Service

The PPoM Service allows secure online monitoring on e-health applications. It provides privacy-aware monitoring APIs to online applications so that the administrators can embed the APIs in the webpages of their applications. Note that the privacy-aware APIs enables them to specify the type of data to be monitored, including types of health-related and

 ISBN: 978-1-61208-511-1

HIPAA-related data that are defined in the HIPAA Profile [5]. The APIs embedded in an e-health application collect data based on a patient's user policies, not an application's policies. It means that the privacy-aware APIs do not collect data if a patient prefers not to disclose. By doing so, the PPoM Service provides a way to protect a patient's privacy from indiscriminate monitoring.

Monitoring data collected contains general usage data such as *device category*, *operating systems*, *event*, and *time* and also user data including health-related data. All collected data are encoded in JavaScript Object Notation (JSON), a lightweight data interchange format, and sent by a patient's web browser to the PPoM Service server. The structure of a JSON monitoring data is shown in Fig. 2.

[ELEMENT_ID|ELEMENT_PATH] [EVENT_TYPE] [TIME] [DATA_ TYPE] [DATA] [DEVICE_INFORMATION]

- *ELEMENT ID*: It is a unique ID of a HTML element.
- *ELEMENT_PATH*: In case of dynamic webpages, a path from the root element is used as an ID if an element does not have ID. The path is unique for each element.
- *EVENT_TYPE*: It denotes that a type of an event occurred. The set of event types are as follows: {*entering a page*, *leaving a page*, *clicking an element*, *filling an element*}.
- *TIME*: It denotes the occurring time of an event
- *DATA_TYPE*: It is a type of monitoring data and must be specified based on the data types in the P3P data schema and the HIPAA Profile [5].
- *DATA*: It is the value of the monitoring data.
- *DEVICE_INFORMATION*: It includes a device's *category*, *operating system*, *language*, and *browser information*.

Figure 2. The structure of a JSON object for monitoring data

Let's assume that there is the *Blood Type* textbox, which its HTML code is shown in Fig. 3 (a). A patient enters "*A*" in the *Blood Type* textbox. The monitoring data on that textbox is then created as shown in Fig. 3 (b). At this time, the values of *EVENT*, *TIME*, and *DEVICE_INFORMATION* are automatically collected by JavaScript Built-in functions. Note that *blood type* is one of the health data type defined in the *Health* data schema of the HIPAA Profile. Before sending the monitoring data to the PPoM service server, a PPoM browser encodes the raw monitoring data as a JSON object as shown in Fig. 3 (c).

Fig. 4 illustrates the pseudo codes that enable the PPoM monitoring service on the server side (the PPoM Service server) and the application side (an e-health application). The `monitor` JavaScript function described in Fig. 4 (a) must be embedded in webpages of a PPoM-enabled e-health application prior to monitoring. When a target event is occurred, the `monitor` function captures and creates monitored data as a JSON object. Towards this goal, the function gathers necessary information by using JavaScript built-in functions and properties. Then, it invokes the `jQuery.ajax` function to communicate with the server-side scripts (the `receiveData` function shown in Fig. 4 (b)). The `jQuery.ajax` function converts a JSON object into a string and sends the string to the PPoM Service server through the HTTP POST method. When receiving a JSON string, the `receiveData` PHP module in the PPoM Service server converts the string into a JSON object and then stores monitoring data in the object in its database.

```
<input id="bloodtype" type="text" data-type
="health.bloodtype"/>
```
a) HTML Representation

```
ELEMENTID: bloodtype,
EVENT_TYPE: TEXTINPUT,
TIME: 2016-07-15T12:45:07,
DATA_TYPE: health.bloodtype,
DATA: Type A,
DEVICE_INFORMATION:DESKTOP(DEVICE CATEGORY),
WINDOWS(OS), ENGLISH(LANGUAGE),FIREFOX(BROWSER)
```
b) Raw Monitoring Data

```
{
 "ELEMENT_ID": "bloodtype",
 "EVENT_TYPE": "TEXTINPUT",
 "TIME": "2016-07-15T12:45:07"
 "DATA_TYPE": "health.bloodtype",
 "DATA": "Type A",
 "DEVEICE_INFORMATION":
  {     "DEVICE_CATEGORY": "DESKTOP",
        "OS": "WINDOWS",
        "LANGUAGE": "ENGLISH",
        "BROWSER": "FIREFOX"          }
 }
```
c) JSON Object of the Raw Monitoring Data

Figure 3. Examples of monitoring data.

```
function monitor(object, event) {
  var monitoredData <= ID, EVENT_TYPE,
TIME, DATA_TYPE, DATA, DEVICE_INFORMATION
from the parameters object and event;
  jQuery.ajax({
    type: "post",
    url: "/PPoM/monitoring.php",
    data: JSON.stringify(monitoredData),
    contentType:"application/json; ",
    dataType: "json"                  });
}
```
a) JavaScript Function on the application side

```
function receiveData($monitoredData) {
  $object = json_decode($monitoredData);
  $link <= connection to database;
  $sql <= create INSERT query to store the
      monitored information ($object)
  mysqli_query($link, $sql);
}
```
b) PHP function in the PPoM service

Figure 4. Pseudo codes of the PPoM monitoring service on the application side (an e-health application) and the server side (the PPoM service).

B. PPoM Browser

The PPoM Browser provides three ways to protect user privacy. First, it enables a non-IT patient to intuitively specify his/her privacy preferences. Through its user-friendly interfaces shown in Fig. 5, a patient can define three different levels of policies, the General Policies (GP), the Application-specific Policies (AP), and the Page-specific Policies (PP). First, the General Policies describe a patient's general

 ISBN: 978-1-61208-511-1

preference regarding data sharing. An Application-specific Policy is applied to an online application and it affects across webpages in an application while a Page-specific Policy is applied to a particular webpage of an application. To define a GP, a patient needs to specify data types that a patient allows or disallows to be monitored across different applications. To specify a AP or a PP, a patient needs to enter the url of an application or a webpage. If two or more policies conflict, then the most specific policy takes precedence. Note that the prototype of the PPoM Browser we developed currently supports the Application-specific policies and the Page-specific polices.

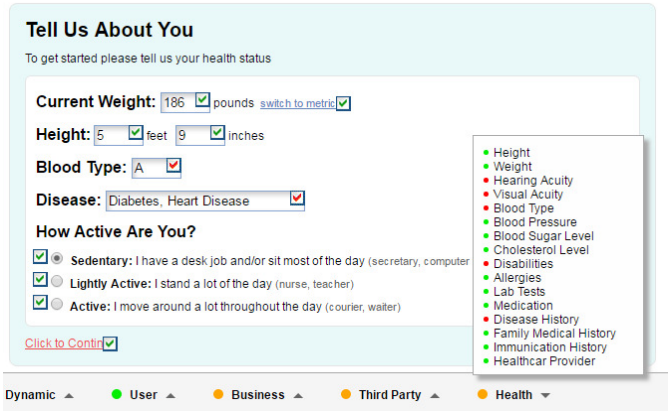

Figure 5. A screenshot of the user interface of the PPoM browser that displays all data being monitored when a user turns on the privacy-preserving mode.

Second, it displays all data being monitored when a patient uses an e-health application. Although a patient agrees to an application's policies, it is critical to verify the application's compliance with the mutually agreed policies. By turning on the privacy-preserving mode of the PPoM browser, a patient can easily figure out that what usage/user data are being monitored and protect data from unwanted monitoring without any IT knowledge and skills. An example use of the developed PPoM browser is shown in Fig. 5. The browser displays which data is being monitored and who is the recipient of the data by using different-colored check marks. The *red* check marks mean that the data is protected and there is no recipient. The *orange* check marks mean that only the first party (for example, an e-health application that a patient is using) is receiving the data. The *green* check marks mean that third parties (for examples, other healthcare providers referred by the first party, advertisement companies, and payment companies) are also receiving monitoring data. A summary of monitoring data, including not only general usage data (such as *HTTP* data, *Clientevents*, and *Cookies* types of *Dynamic* data schema in P3P) and user data (*name*, *bday*, and *location* of *User* data schema in P3P) but also health-related data (*blood type*, *disabilities*, and *disease history* of *Health* data schema in the *HIPAA profile*) on a webpage is displayed in the status bar.

Third, it enables a patient to stop monitoring on a specific page or web object if he/she finds out fraud activities that are against the mutually agreed policies. Towards this end, the PPoM Browser first renders all clickable web objects -- such as buttons and objects handled by JavaScript click-event handler-- and input HTML elements, such as textbox and checkbox. It then creates checkboxes for each of them. By changing the colors of checkmarks, a patient can easily control data sharing. At this time, a patient has multiple options for selecting objects: 1) select all clickable and input elements, 2) select all clickable elements, 3) select all input elements, 4) select an individual (clickable or input) element, or 5) select *None*.

To block monitoring, a PPoM browser needs to generate and run JavaScript codes based on a patient's selection in real-time. Before explaining the blocking process further, let's assume that the main functionalities of an e-health application does not depend on JavaScript. To block online monitoring on a particular web element, a PPoM browser disables JavaScript event handlers that are associated with the selected web elements, and in turn, a monitoring JavaScript using those handlers will be disabled. For example, if a patient marks the *Blood Type* textbox in *red* color to block monitoring on that textbox, then the following JavaScript code is generated to disable the PPoM monitoring JavaScript on the *Blood Type* textbox (The ID of the textbox element is "BLOODTYPETXT"): `$("body").off("keyup keypress change click blur", "#BLOODTYPETXT")`. The generated code then removes five event handlers from 'BLOODTYPETXT' element by invoking jQuery `off` function. When the blocking code runs, none of monitoring services obtain data from that textbox.

C. PPoM Tools (PPoMT)

The PPoMT is a toolkit on the server side, which enables non-IT healthcare providers to generate application policies and monitoring codes for their own e-health applications through intuitive user interfaces and also easily upgrade their existing applications into a PPoM-enabled e-health application. The detailed explanation for each component in the PPoMT is below.

1) In-Page Selector

The goal of the In-Page Selector is to generate modified webpages that shows selectable HTML elements on webpages so that an administrator of an e-health application can select usage/user data to be monitored and policies corresponding to each data, each page, or an entire application. When an administrator selects a web element or a group of elements, another window is popped up to specify a privacy policy about the selected element(s). As shown in the Fig. 6, a non-IT healthcare provider can create a P3P-based application policy which deals with health data and HIPAA regulations. Towards this end, a set of data and policies that are selected by an administrator is delivered to the Privacy Policy Generator and the Monitoring Code Generator.

 ISBN: 978-1-61208-511-1

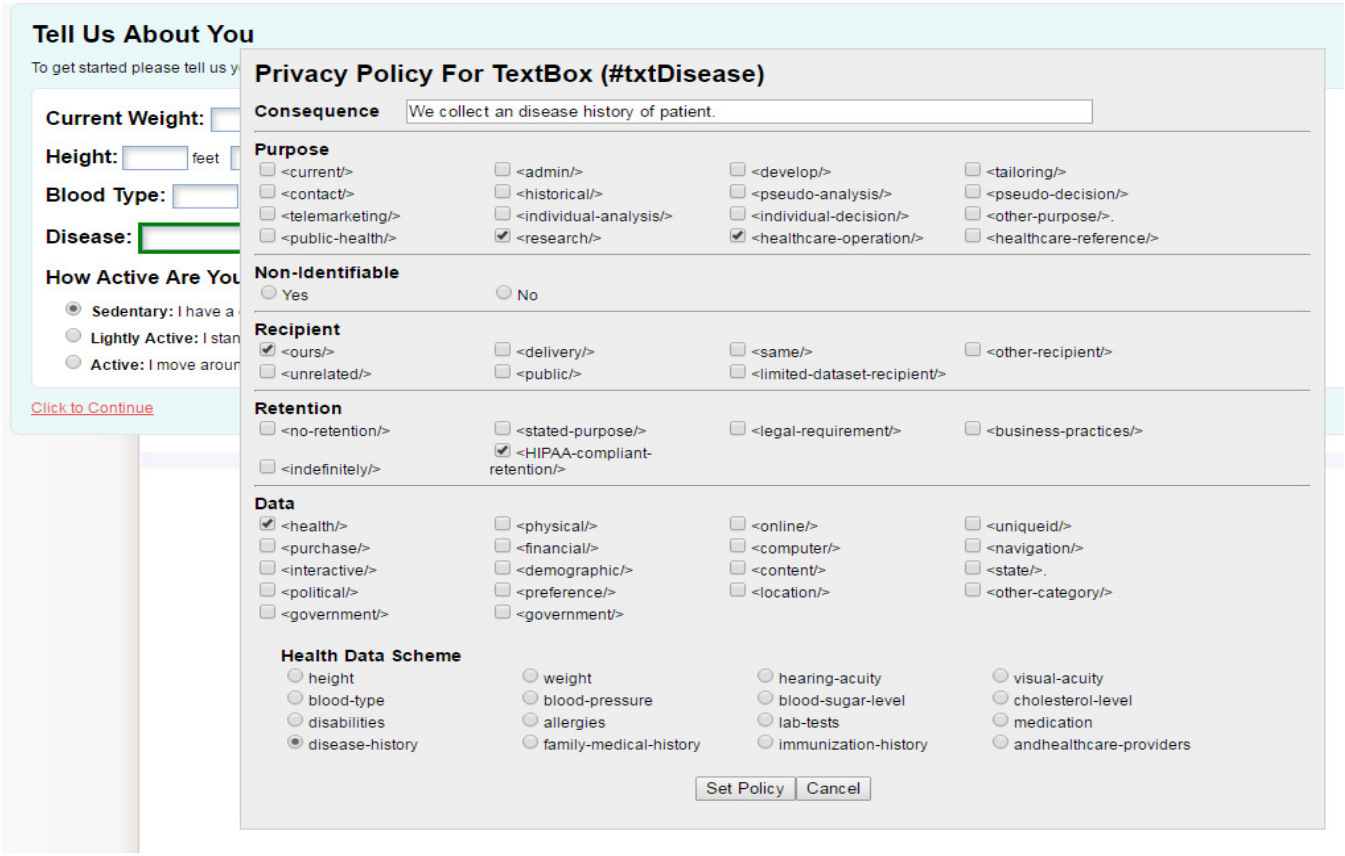

Figure 6. An example of the screenshot of the PPoMT.

2) Privacy Policy Generator

The Privacy Policy Generator generates an application's policies in the HIPAA-Compliant Privacy Policy Language [5] based on an administrator's selection. Not only an administrator of an e-health application but also a patient can use the Privacy Policy Generator to specify a patient's user policy for a specific e-health application. To do so, a patient needs to enter the url of an e-health application and select *User Policy Generation* rather than *Application Policy Generation*. If the *User Policy Generation* is selected, the Policy Generator creates APPEL policies employing the HIPAA profile.

3) Monitoring Code Generator

When receiving a set of data to be monitored and relevant policies, the Monitoring Code Generator produces privacy-aware monitoring codes for an e-health application. Toward this goal, it first checks if each selected element has an ID. If an element doesn't have an ID, it assigns a unique element ID and generates JavaScript code using the assigned ID. Depending on the type of an application, a *static* application or a *dynamic* application, a generated element ID will be different. If an e-health application is a *static* application that the same HTML code stored in an application's server is delivered to all patients' browsers, the Monitoring Code Generator assigns an absolute ID to an element. Let's assume that a webpage has several textboxes and its HTML code is shown in Fig. 7 (a). In this example, the *Current Weight* textbox has its ID ("`WEIGHT`") but other three textboxes -- the *feet*, the *inches*, and the *Blood Type* textboxes-- do not have their IDs.

If the webpage is a *static* page, then the Monitoring Code Generator automatically creates IDs for three textboxes: "`PPOM-ELEMENT-0001`" and "`PPOM-ELEMENT-0002`" for the *feet* and *inches* of the *Height* textbox and "`PPOM-ELEMENT-0003`" for the *Blood Type* textbox. The generated monitoring code for the *static* webpage is shown in Fig. 7 (b). On the other hand, if an application is a *dynamic* application that the HTML codes are dynamically generated by an application server on demand, a path of an element from a root of a Document Object Model (DOM) object is used as a unique ID because an element's path is unchangeable. The monitoring code for a *dynamic* webpage is shown in Fig. 7 (c). As you can see, except the pre-defined ID of the *Current Weight* textbox, for other three textboxes which do not have IDs, their paths are used to identify each textbox. For examples, "`input[type=text]:nth-of-type(2)`" and "`input[type=text]:nth-of-type(3)`" for the *feet* and *inches* of the *Height* textbox and "`input[type='text']:nth-of-type(4)`" for the *Blood Type* textbox. Note that the PPoMT uses the PPoM Service so privacy-aware monitoring script code is generated as default, but it is possible to use different monitoring services such as Google Analytics.

 ISBN: 978-1-61208-511-1

```
<body>
Current Weight:<input id="WEIGHT" type="text">lbs.
Height: <input type="text">feet
        <input type="text">inches
Blood Type: <input type="text">
<input id="BUTTON1" type="submit" value="Submit">
</body>
```

a) A partial HTML code of an e-health application shown in Fig. 5

```
<body>
Current Weight: <input id="WEIGHT" type="text">
          lbs.
Height:<input id="PPOM-ELEMENT-0001" type="text">
          feet
      <input id="PPOM-ELEMENT-0002" type="text">
          inches
Blood Type:
      <input id="PPOM-ELEMENT-0003" type="text">
<input id="button1" type="submit" value="Submit">
</body>

<script>
   $("#WEIGHT").change(function() {
      monitor($(this), "change");
   });
   $("#PPOM-ELEMENT-0001").change(function() {
      monitor($(this), "change");
   });
   $("#PPOM-ELEMENT-0002").change(function() {
      monitor($(this), "change");
   });
   $("#PPOM-ELEMENT-0003").change(function() {
      monitor($(this), "change");
   });
   $("#BUTTON1").click(function() {
      monitor($(this), "click");
   });
</script>
```

b) HTML code converted by the PPoMT in case of a static application

```
<script>
$("#WEIGHT").change(function(){
     monitor($(this), "change");});
$("input[type='text']:nth-of-
type(2)").change(function()
   {  monitor($(this), "change");});
$("input[type='text']:nth-of-
type(3)").change(function()
  {  monitor($(this), "change");});
$("input[type='text']:nth-of-
type(4)").change(function()
  {  monitor($(this), "change"); });
$("#BUTTON1").click(function(){
     monitor($(this), "click"); });
</script>
```

c) Monitoring JavaScript code generated in case of a dynamic application

Figure 7. Examples of monitoring code generated by the PPoMT.

4) Application Converter

This component produces a PPoM-enabled application by inserting monitoring codes generated by the Monitoring Code Generator into DOM objects for each webpage in an application. Depending on a type of an application, the conversion process would be different. In case of a *static* application, the Application Converter can systematically generate the updated HTML code. If an application is however a *dynamic* application, the PPoMT provides monitoring script code only and then an administrator should manually insert the generated monitoring code into the server-side program that dynamically creates webpages.

IV. EVALUATION RESULTS

To test the usability of the proposed PPoM framework, we first develop two types of sample e-health applications that each has ten webpages containing different numbers of monitoring elements without IDs and a prototype of the PPoM framework, including the PPoM service, the PPoM browser, and the PPoMT. Then, we test the performance of the PPoMT according to the evaluation plans described in [4].

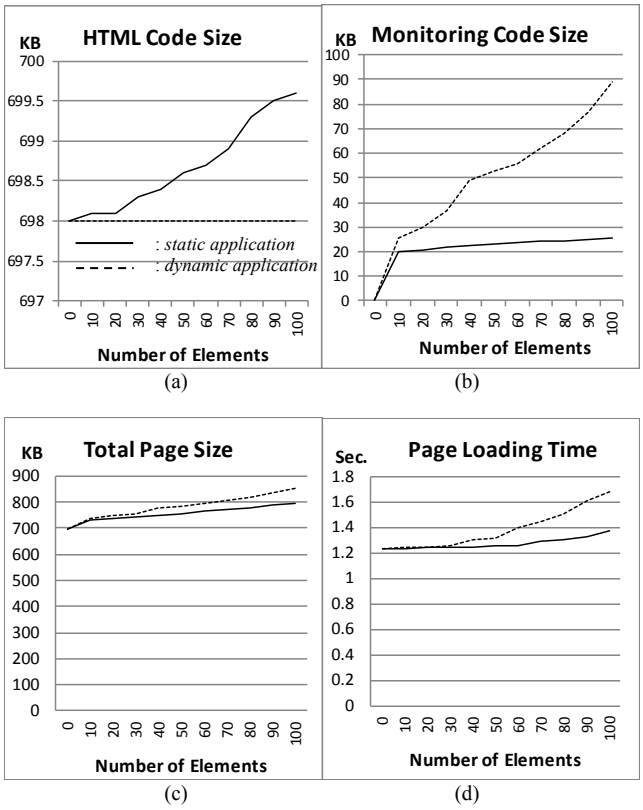

Figure 8. Evaluation Results.

As shown in Fig. 8, the size of HTML code that generated by the Application Converter for *dynamic* webpages is zero since the PPoMT will not generate HTML code for a dynamic application while the size of HTML code for *static* webpages increases linearly as the number of the monitoring elements increases.

In case of *static* webpages, the size of the generated monitoring code remains steady once it reaches a certain size even though the number of monitoring elements increases linearly. However, the size of monitoring code for *dynamic*

 ISBN: 978-1-61208-511-1

webpages linearly increases according to increase of the number of monitoring elements (See Fig. 8 (b)). This is because specifying paths from root is costly, especially for complex web pages.

As you can see in Fig. 8 (c) and (d), in case of *static* webpages, the number of monitoring elements does not affect the size of converted webpages that produced by the PPoMT and also the page loading time. However, in case of *dynamic* webpages, the increase in the number of monitoring elements affects the page loading time. If a *dynamic* page is complex, the loading delay becomes a big obstacle. It is one of our challenges to find out a way to minimize the loading delay caused on dynamic webpages.

TABLE I. FAILURE RATIO IN THE PPOM SERVICE

	Monitored	Not Monitored
Allowed	(a) 4,897	(b) 345
Not Allowed	(c) 0	(d) 3,477

TABLE II. SUCCESSFUL BLOCKADE RATIO IN THE PPOM BROWSER

	Sent	Blocked
Allowed	(e) 5,242	(f) 0
Not Allowed	(g) 0	(h) 3,477

To evaluate the privacy protection of the PPoM Browser and the PPoM Service, we produce five hundreds of different sets of patients' privacy preferences (i.e., allow or disallow monitoring on particular web elements) and activities on a sample static application (i.e., navigating webpages, clicking buttons, or entering data in input elements). Using the sets of user policies and activities, we test the PPoM browser and the PPoM Service. To evaluate the privacy protection on the prototype of the PPoM framework, we measure two factors: the failure ratio ($c/(a+c)$ in Table I) and the successful blockade ratio ($h/(g+h)$ in Table II).

The failure ratio evaluates privacy protection on the server side (the PPoM Service server) and the successful blockade ratio evaluates protection on the client side (the PPoM Browser). As shown in Tables I and II, none of user/usage data that patients do not allow to be monitored was captured by the PPoM Service and none of the unauthorized data was sent from the PPoM Browser. However, as shown in Table II, we found some data loss in the PPoM Service server. The PPoM Browser sent 5,242 data ($a+b$), but the PPoM Server received only 4,897 data (a). It may be caused by heavy load of transaction or overheads for checking application and user policies. To figure out the source of the resulting data loss, we need to investigate more in the future.

V. CONCLUSION

Privacy protection is critical for widespread use of e-health applications. Without proper methods for the privacy preservation, people may keep hesitating to use e-health applications, even though e-health applications help them access healthcare services in easy and convenient way at the reduced cost. To address the privacy issue, we develop the prototype of the PPoM framework that protects user privacy in both the application side and the user side. To prove the performance and the usability of the developed prototype, we present the evaluation results. Towards our ultimate goal, however, we need to complete the following tasks in the future:

- Research a way to reduce data loss ratio on the PPoM service.
- Investigate a method to reduce the size of the monitoring codes generated by the PPoMT, especially for dynamic e-health applications.
- Field tests of the prototype with actual e-health applications.

REFERENCES

[1] J. R. Mayer and J. C. Mitchell, "Third-Party Web Tracking: Policy and Technology," Proc. IEEE Symp. on Security and Privacy (SP '12), IEEE Press, pp. 413-427, 2012.

[2] M. Bilenko and M. Richardson, "Predictive client-side profiles for personalized advertising," Proc. ACM SIGKDD conf. on Knowledge discovery and data mining (KDD '11), ACM New York, pp. 413-421, 2011.

[3] A. McDonald and L. F. Cranor, "Beliefs and Behaviors: Internet Users' Understanding of Behavioral Advertising," TPRC 2010 Social Science Research Network (SSRN), August 16, pp. 1-31, 2010, Available from http://ssrn.com/abstract=1989092 [retrieved: September, 2016].

[4] Y. Jung, "Toward Usable and Trustworthy Online Monitoring on e-health Applications," International Journal On Advances in Life Sciences, vol. 8, numbers 1 and 2, pp. 122-132, June, 2016

[5] Y. Jung and M. Kim, "HIPAA-Compliant Privacy Policy Language for e-health Applications", Proc. the 6th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare, September, London, United Kingdom, in press.

[6] P3P 1.1. http://www.w3.org/TR/P3P11/ [retrieved: September, 2016].

[7] A P3P Preference Exchange Language (APPEL) version 1.0. https://www.w3.org/TR/P3P-preferences/ [retrieved: September, 2016].

[8] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "An XPath-based preference language for P3P," Proc. the 12th international conference on World Wide Web (WWW '03), ACM, New York, pp. 629-639, 2003.

 ISBN: 978-1-61208-511-1