# Yale University
# Department of Computer Science

P.O. Box 208205
New Haven, CT 06520–8285


**Privacy from Untrusted Web Servers**


Robert Fischer[1]
*Harvard University*

Margo Seltzer[2]
*Harvard University*

Michael Fischer[3]
*Yale University*



YALEU/DCS/TR-1290
May 19, 2004



[1]Email: rfischer@rics.bwh.harvard.edu
[2]Email: margo@eecs.harvard.edu
[3]Email: fischer-michael@cs.yale.edu

# Privacy from Untrusted Web Servers

Robert Fischer[*] Harvard University

Margo Seltzer[†] Harvard University

Michael Fischer[‡] Yale University

**Abstract**

Many potentially useful web applications require use of user private data. In the typical client-server paradigm, the server must be trusted to respect the user's privacy, for the user's private data is sent to the server for processing. The *Clilet* system is a web application architecture and protocol that guarantees user privacy even in the face of untrusted servers. User private data never leaves the client's domain of trust, even though the code and additional data needed to process it comes from the untrusted server. The Clilet architecture is described along with a security analysis. A fully functional prototype implementation and sample applications are also described.

## 1 Introduction

Alice's company requires each of its employees to use a web-based calendar application for scheduling work-related appointments. This application runs on a web server hosted by her company and enables management to schedule meetings at times that minimize employee conflicts.

Alice is willing to use this application for work-related appointments; however, she does not wish others to see the details of her personal engagements, which she regards as private. She is perfectly willing to disclose those times at which she is unavailable for personal reasons, just not the details of those reasons. She therefore keeps personal engagements on a separate calendar stored on her laptop. To schedule a new appointment, she must consult both calendars in order to avoid potential time conflicts. She also must make two entries when scheduling a personal appointment, once on her laptop with full details, and another on the company's calendar in order to block out the time as "personal." This latter step is necessary in order to prevent the company from scheduling events when she is not free. Needless to say, she finds this manual integration of data from two sources and requirement of double entry to be cumbersome and error-prone. Because she forgets to block out her personal time on the company's calendar, the company's calendar application fails to achieve its goals of minimizing conflicts with official meetings.

The company's IT staff, realizing that the calendar application is not working as intended, implements a new feature that allows Alice to identify each appointments as being either "work-related" or "personal". When management accesses Alice's calendar, details for all appointments marked "personal" are suppressed. Unfortunately, this still does not allay Alice's concerns, for she neither trusts the IT staff with her private personal data, nor does she trust that they will not disclose it to management upon request.

The *Clilet* system described in this paper is designed to address Alice's concerns. A Clilet browser provides Alice with an integrated view of private data from her laptop with public data from her company's Clilet server, allowing her to view a single "virtual" calendar that contains both her public and private appointments. It also prevents private data from being returned to the server while allowing the public information to go back unhindered.

[*]Email: rfischer@rics.bwh.harvard.edu

[†]Email: margo@eecs.harvard.edu

[‡]Email: fischer-michael@cs.yale.edu

While one could easily imagine how to implement a specialized web browser tailored to this particular application, actually doing so would be a major task, and assurance that sensitive data could not leak back to the server would be difficult to achieve. Moreover, if Alice's IT department produced such a program, Alice would have little reason to trust that it worked as advertised. So there are two separate issues here: producing code to run on Alice's computer that prevents the leakage of sensitive information, and providing Alice with reasons to trust that code.

Clilets address both concerns. The Clilet system is a general web-like client-server framework for computing with a user's private data while maintaining its privacy. The part of the system that runs on Alice's computer is called the Clilet browser. In order to solve any particular task, the browser runs a piece of server-provided mobile code called a *clilet* that specifies how to carry out the task.

Unlike many other approaches to such problems, the clilet code itself does not have to be trusted. Alice's privacy assurance comes from the Clilet browser itself. As long as she trusts that her Clilet browser correctly implements the Clilet protocol, the privacy of her data is guaranteed, regardless of what the clilet code might attempt to do. Therefore, Alice has no reason *not* to use the Clilet application provided by her company for scheduling appointments.

### 1.1 Summary

In summary, the Clilet system is a system for building web-like applications that allow for private data stored and managed by the client. The system guarantees that the private data is never disclosed to the server. To accomplish this, the web protocols are extended so the server sends the browser pieces of mobile code known as *clilets*. Clilets are executed by the Clilet browser. They access private data and produce HTML output, which the Clilet browser displays, integrating private and server-provided data. Clilets are able to process private data as needed, eliminating the need to disclose that data to the server. Care is taken by the browser to safeguard against malicious clilets.

A number of application domains exist today in which web applications are technically desirable but are often not used due to privacy concerns: personal finance, on-line tax preparation, and personalized health information to name a few. The Clilet system allows the construction of web-like applications for these domains that guarantee users' private data remains private.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 details the trust model under which the Clilet system runs, providing a Clilet-enabled solution to Alice's problem. Section 4 introduces the Clilet protocol and gives an overview of client-server interaction along with a walk-through of a client-server interchange. Section 5 focuses on the steps taken by the Clilet browser to maintain the privacy of private data. It goes on to discuss possible threats to that privacy and what is done about those threats. Section 6 describes the kinds of applications for which the Clilet system is and is not well-suited, giving a few examples. We discuss possible avenues of future research in section 7 and conclude in section 8.

## 2 Related Work

The Clilet system is a general-purpose client-server system that allows for private data. Private data is kept secret from the server and no trust or trusted third party is required between client and server. As far as we know, the Clilet system is the only system to allow for private data in this way; however, it is related to a number of other systems.

### 2.1 Private Information Retrieval

*Private information retrieval* refers to cryptographic schemes that store private data on untrusted servers and allow the client to query that data while keeping it secret from the server [12, 7]; however, practical private information retrieval schemes for large amounts of data have not yet been demonstrated.

Although the Clilet system solves a similar problem, it does so in a very different way. Rather than relying on cryptographic unbreakability assumptions, the Clilet system prevents an untrusted server from learning about private data by not sending those data to the server to begin with.

### 2.2 Decentralized Label Model

The *Decentralized Label Model* and related work [10, 11] focus on providing information flow guarantees in a distributed system. Data are tagged with labels; these labels are used by the system to enforce information flow policies defined by the owner of the data. Awareness of information flow labels is built into *Jif*, a security-typed Java variant with support for static information flow control.

One can think of numerous uses for the information flow tools provided by the Decentralized Label Model and Jif. Among other things, the system allows for software from party $A$ to operate on data from party $B$ in and environment of mutual distrust. However, the Decentralized Label Model addresses a somewhat different problem from the Clilet system: it assumes a mutually trusted third party, whereas the Clilet system does not.

### 2.3 Multi-Level Systems

The Clilet system uses asynchronous clilet execution (Section 5.1) to control the flow of private data within a clilet. Multi-level system [1] have been used to achieve a similar effect in multi-user, multi-tasking system. Covert channels are common in multi-level systems [9]; in our case, with only one thread of control, we are able to provably prevent them [10].

### 2.4 Java Applets

Other approaches exist to safeguard data privacy under different sets of assumptions. The Java Applet, for example, is a piece of untrusted code designed to run in a web browser. It is prevented from damaging the local system; however, any data to which it is given access can be transmitted to the server. It is therefore similar to the public clilet segment in terms of privacy.

## 3 Trust Model and Example

The Clilet system assumes the following trust model:

- The client and server are mutually distrustful, with no mutually trusted third party.
- The computation requires data from the client and from the server; if data from only one or the other were needed, a client-server application would not be necessary.
- The client wishes to prevent the server from deducing some of the data it brings to the computation. This data is known as *private data*. Similarly, the server is trying to deduce the contents of the private data.
- The client is required to trust the Clilet browser. This software, like the client's operating system, was produced independently of the server.
- The Clilet application application consists of parts that run on the server and parts that run on the client. Both are produced by the server. Parts that run on the client are known as clilets.

The above requirements introduce a paradox. Since private data and data from the server are both required for the computation, the clilet code must be able to communicate with the server and access private data. However, this code is not trusted. Untrusted code is commonly sandboxed, given only certain permissions. What will those permissions be? In general, code that can access private data and communicate with the server will be able to communicate that data to the server. Therefore, it seems there is no way to simultaneously safeguard private data and allow the clilet to do its job of computing with data obtained from the client and the server. We solve this seeming paradox by using a more powerful sandboxing model (see Section 5.1).

### 3.1 Clilet User Experience

We designed the Clilet system so that the Clilet applications look and feel like web applications. Note that private data is not possible in today's web applications because *all* data processed by the application is visible to the server. Therefore, the Clilet system can be thought of as introducing an extended web application protocol. We have developed a fully secure, fully functional Java-based prototype of the Clilet system.

The Clilet browser, akin to a web browser, is a trusted piece of on the client machine used to connect to a Clilet server. To users sitting at a Clilet browser, a Clilet session works much like an HTTP/HTML session. Users are presented an HTML page with which they interact. When a user clicks on an anchor or form submit button, the Clilet browser initiates a round-trip communication with the server, called an *interchange*; the effect of the interchange is to display a new HTML page in the browser. Sessions are managed through the use of session keys shared between browser and server.

From the user's point of view, there is one major difference between a Clilet and an HTTP/HTML application: every HTML element on the screen, including form elements, is designated by the server as *public* or *private*; the Clilet browser makes this distinction clear to the user through user interface cues. Data entered into public form elements is passed along to the server, the same as with HTTP/HTML. However, the Clilet browser ensures that data entered into private form elements is *never* revealed to the server, although it may be stored client-side.

### 3.2 Example with Clilets

The IT staff of Alice's company could address Alice's concerns by building the shared calendar application with the Clilet system. Alice logs into that application to view her day's appointments. Public and private appointments, generated by the public and private clilet segments, are shown in a distinctive fashion. Alice is looking for a time for a doctor's visit this afternoon. She notices 13:00 is free.

Alice clicks on a button marked "New Private Appointment" and is brought to a screen asking her for more details. It contains a form with public and private form elements, visually distinguishable from each other. The date, time and duration of the appointment are being requested in public form elements; all other information is being requested in private form elements.

Alice fills out the requested information and clicks *Add*. The time and duration of her doctor's visit are sent to the server; all other data she filled in is stored locally. When Alice next clicks on "View Calendar," she sees the complete details of the private appointment she just made, as well as details of other appointments loaded from the Clilet server.

## 4 Clilet Protocol

The workings of the Clilet system protocol are similar to HTTP with one important difference. Rather than returning HTML to the browser, the server returns a *clilet* — a kind of mobile agent that includes data and executable code. The clilet, when run, produces HTML, which is displayed to the user. It also reads and writes in the private data store.

Clilets are sent to the Clilet browser in two pieces, the *public clilet segment* and the *private clilet segment*, which are given different privileges when run. The public clilet segment is given access only to public data, and it generates the bulk of the HTML. The private clilet segment is responsible for processing private form

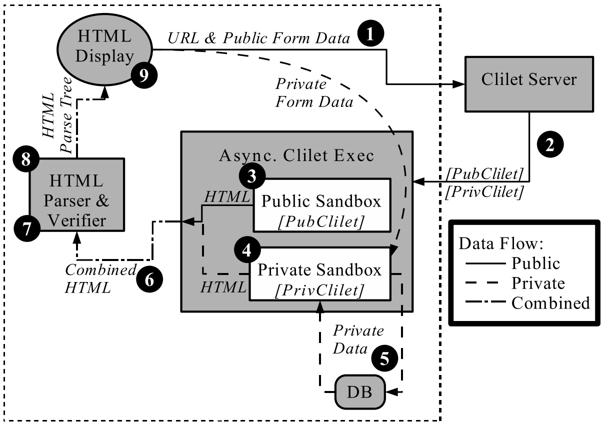


Figure 1: The major components of the prototype Clilet browser, and how they interact on a client-server interchange. Arrows indicate public, private and combined data flows. Numbers indicate the order in which actions take place. All components within the dotted box on the left are part of the Clilet browser.

elements, for accessing the private data store, and for generating HTML that will solicit or display private data for the user. It is *not* allowed to write HTML that could result in the transmission of data to the server.

The Clilet system uses an XML-based variant of HTML called *Clilet HTML* [4]. It is similar to XHTML 1.1 [2], with some changes made to accommodate public and private data within one document. As mentioned above, only HTML that conforms strictly to the XHTML 1.1 grammar is allowed — this enables us to reason clearly about privacy. It would be possible to accept mal-formed HTML as well, as long as that HTML is parsed in a consistent, well-defined manner.

## 4.1 Clilet Browser

The Clilet browser is responsible for communicating with the server and user and for running clilets in a secure fashion. It is conceptually divided into a number of interacting parts (Figure 1), which we will describe more fully below.

The *HTML Display* is responsible for displaying HTML, interpreting user input and communicating with the server.[1] *Asynchronous clilet execution* is a technique used by the Clilet browser to run a clilet — it involves the use of a *public sandbox* and a *private sandbox*, giving different rights to the (untrusted) code being run within. The *private data store* is used to store private data. The *HTML Parser* is responsible for parsing HTML into a parse tree. The *HTML Verifier* checks the HTML for privacy violations.

## 4.2 Sample Interchange

A typical client-server interchange works as follows (Figure 1):

1. Upon prompting by the user, the HTML Display sends an HTTP request to the Clilet server.
2. The server processes the request in an application-dependent manner. When it is ready, it returns public and private clilet segments to the browser.
3. The public clilet segment is executed, isolated from the world inside a *public sandbox*. It is allowed to call functions in the private segment; however, these functions are not allowed to pass information

[1] In our prototype, HTML Display functionality is implemented mostly by the standard web browser.

back or call functions in the public clilet segment. In order to enforce this rule, calls to the private clilet segment are queued by the Clilet browser, only to be run upon completion of the public clilet segment.

4. Queued private clilet segment calls are executed inside a separate *private sandbox*. Data from private form elements associated with the interchange are made available at this time. The public and private sandboxes implemented in a straightforward manner using the Java 2 Platform Security [8].
5. In the course of execution, the private clilet segment reads and writes in the private data store.
6. Output from the public and private clilet segments is arranged into a complete HTML document consisting of alternating public and private HTML segments (determined by whether the output was produced by a public or private clilet segment). The output of each private function call (a *private HTML segment*) is inserted into the output of the public clilet segment (a *public HTML segment*) at the point that the public clilet segment queued the private function.
7. The HTML Parser parses the combined public/private HTML output, producing a parse tree. Each node in the tree corresponds to either a begin/end tag pair, or to a block of text. Each node is labeled as public or private, depending on whether its textual representation was written by the public or private clilet segment. Our HTML Parser is somewhat different from a standard XML/HTML parser because it is aware of public and private HTML segments. If the HTML cannot be parsed or annotated — for example because a tag was written partially by the public clilet segment and partially by the private clilet segment — a fatal error is thrown and the interchange ceases.
8. The HTML Verifier ensures that no security rules are violated by the HTML represented by the parse tree. It checks, for example, against hyperlinks annotated as *private*.
9. The HTML Display formats the resulting parse tree on the user's display and solicits user interaction.

# 5 Protecting Privacy

The Clilet system as described above requires the client-side execution of (untrusted) code originating from the server. That code might try to damage the client system or transmit private data back to the server. It is the job of the Clilet browser to guard against these attacks. The job of protecting a local system from untrusted code is a well-studied topic. We use the Java Security framework to prevent a clilet from executing dangerous code, e.g. reformatting the filesystem — these are the same mechanisms used by classical browsers to execute applets.

A more interesting issue is how one can prevent the clilet from transmitting private data to the server. Only the private clilet segment is given direct access to private data. Therefore, if we can prevent the private clilet segment from transmitting information to the server, we have achieved our objective. There are four ways a private clilet segment might try to do this. They are listed below with the section in which we describe how we safeguard it provided in parentheses.

1. It could transmit private data to its calling public clilet segment, which then relays that data to the server (Section 5.1).
2. It could write something in its output that causes the HTML Display to transmit data to the server — for example, it could try to put an anchor into a private HTML segment (Section 5.2).
3. It could vary its running time, thereby varying the overall round-trip time observable by the server. This could be used to convey arbitrary private data (Section 5.3).
4. It could try to trick the user into revealing private data to the server (Section 5.4).

We will now discuss each of these potential attacks and the mechanisms used to thwart them.

### 5.1 Asynchronous Clilet Execution

The simplest way for private clilet segments to provide data to public clilet segments would be via explicit return values from function calls. We remove this possibility by forbidding return values from private clilet segment execution. More fundamentally, we need to show that there is no way for private clilet segments to implicitly pass information to their calling public clilet segment: that the behavior of the public clilet segment must be *oblivious* to the actions of the private clilet segments.

This is easily proven because of our *asynchronous execution* model. Private clilet segment code is not executed right away when the public clilet segment calls a private segment function. Instead, the calls are queued, then run only *after* the private clilet segment finishes executing. The result is the same as if the private clilet code had been run right away; however, it is now easy to prove that the behavior of the private clilet segment cannot affect the public clilet segment.

### 5.2 HTML Integrity

The private clilet segment is allowed to write to the HTML output stream because it needs to be able to display private data to and solicitg private data from the user. For example, private calendar information must be displayed when the user asks for it.

However, certain kinds of HTML could result in the compromise of data privacy. For example, an anchor tag causes data to be sent to the server when the user clicks on the anchor. An anchor tag from the private clilet segment (a *private anchor tag*) could leak private data, and therefore must be prevented by the HTML Verifier.

This is a simple task, given a parse tree representation of the HTML document. However, other types of HTML must also be prevented as well. For example, an `<img>` tag could cause the browser to request an external image from the server; this is known as a *secondary* interchange. The HTML Verifier must address *all* possible HTML-based privacy breaches, since even one missed vulnerability could be exploited.

We address this issue with a systematic tag-by-tag construction of HTML's semantics [4]. The semantics, available in simple tabular form, allows us to precisely determine *exactly* which tags might leak private data and ban them (see Appendix A & B). Since web browsers must understand HTML semantics and manage communication with the server, the task of building the HTML Verifier is clearly no harder than that of building a web browser.

#### 5.2.1 Preloaded Secondary Interchanges

Prohibiting all tags that spawn secondary interchanges is overly restrictive; the private clilet segment might have a legitimate use for inline images, for example.

Therefore, Clilet HTML provides a way to pre-load secondary interchanges via the `<preload>` tag. This tag, available only to the public clilet segment, initiates a secondary interchange, causing the HTML Display to download the requested resource. When the HTML Display encounters a private node that would otherwise initiate a secondary interchange, it looks for the requested resource (indexed by URL) in the set of preloaded resources specified in the current interchange. If the resource is available, it is used; otherwise, an error is generated. This design trades off efficiency (e.g., on-demand loading) for security (removal of a covert channel).

Since only public `<preload>` tags are allowed, this scheme maintains the privacy of private data. At the same time, it allows private clilet segments to make use of resources downloaded via secondary interchanges.

## 5.3 Covert Timing Channels

By ensuring that the server sees the same view no matter what action is taken by the private clilet segment, the above system prevents the private clilet segment from directly transmitting information to the server. However, the private clilet segment could still vary its running time in an attempt to affect the overall interchange round-trip time, thereby creating a covert communication channel.

This type of covert channel is not a significant threat for many applications because it is noisy and low-bandwidth. The total round-trip time for an interchange can be dozens of seconds, and numerous confounding factors — including the user — add additional delay. These delay will be far greater than any "extra" delay added by the private clilet segment that would still be tolerable to the unsuspecting user. [2]

## 5.4 Tricking the User

We have shown above that the private clilet segment is unable to communicate with the server in any meaningful way. This leaves open the possibility that the clilet might try to trick the user into revealing private information. Clearly, we expect that the user will not put private data — or a function thereof — into public form elements. However, there are other more subtle ways the clilet might try to trick the user.

### 5.4.1 Application Structure

Many web application structures force users to divulge private data simply to use the web site. For example, the IRS web site ( *www.irs.gov*) allows users to download tax forms. It can already deduce something about a user's income based on whether the user chooses to download the 1040, 1040A or 1040EZ form. Astute users could obfuscate their choice by downloading all three; but web applications bent on discovering private information could prevent users from making more than one choice.

The IRS web site is a simple example of this problem. In general, any web application requiring users to make a series of if-then choices based on private data will compromise that data's privacy — even if the data is never explicitly handled by the application. In contrast, an application in which the user's choices are largely oblivious to the data or depend on private data in an unpredictable fashion — such as a public/private shared calendar — do not suffer from this problem.

It is clear that this problem of application structure is a problem for HTTP/HTML as well as Clilet web applications, even in the case (as with the IRS) that no private data was explicitly present. Full investigation of web application structure as a covert channel is outside the scope of this paper.

### 5.4.2 Mis-Labeling

The user chooses what to put into public form elements based on the HTML text surrounding them. Mis-labeled form elements can be used to trick the user, allowing the private clilet segment to divulge private data even if the user provides only public data! Consider a form with two form elements, which we will call $A$ and $B$. These form elements will be used to request name and phone number, which are *not* private data. Textual labels are used to tell the user whether the name should go in $A$ and the phone number in $B$, or vice versa.

If the private clilet segment is allowed to provide these labels, then it can label $A$ as name and $B$ as phone number, or vice versa — depending on the value of a private bit $p$. The server can determine what labels the private clilet segment used by checking whether the value in $A$ looks like a phone number and the value in $B$ looks like a name — or vice versa.

[2] This point addresses covert channels between private clilet segment and server. Covert channels from private to public clilet segment are completely prevented by running the private clilet segment *after* the public.

In order to prevent this problem, the user must trust only labels appearing in public HTML segments — in fact, the user must avoid following *any* directions found on the private part of the HTML page indicating how to fill out the form elements! Clearly, it is difficult to make any automated rule against private-mode HTML text that describes how to fill out a form. To aid the user in avoiding private-mode HTML directions, the HTML Display provides visual cues indicating the public or private nature of every on-screen Clilet HTML element.

### 5.5 Discussion

In this section, we have approached privacy threats by systematically examining the ways in which the private clilet segment might try to communicate with the outside world — and then by showing how the Clilet browser design prevents these attacks. These techniques, when taken in combination, are sufficient to ensure the privacy of private data within one interchange.

In spite of all technical security means, the clilet can still try to trick users in a number of subtle ways. Luckily, experienced users can determine the safety of an application simply by examining it from the outside; no knowledge of the source code is required.

Even so, users are often the weakest security link. We have therefore sought to make security statements under "reasonable" assumptions of user awareness [6]. We assumed that the user does not enter a predictable function of private data into a public form element; nor does the user follow directions given in a private HTML segment. Under these conditions, we showed that any loss of private data is due to the structure of the application. These kinds of privacy problems commonly exist in HTTP/HTML as well as Clilet applications — the IRS web site, for example.

Although Clilet applications requires a heightened level of attention on the part of the end user, they do something that traditional web applications do not do — allow for private data. Whether the user is willing and able to pay attention will depend on the user and the degree to which that user values the privacy of the data being processed. In the case that Clilet applications are not mass-market tools but rather specific applications with a small number of well-trained users, it is quite plausible that users will be trained to take the necessary precautions.

## 6 Appropriate Uses for Clilets

We began with some motivating examples for private data. Now, having described the Clilet system and shown how it maintains privacy, we proceed to describe the kinds of problems we expect Clilets to solve. Although we do *not* expect Clilet applications to replace classical web applications, we *do* expect that they will be seen as the best approach for a certain class of applications.

The Clilet system is most appropriate for application that *combine* data from public and private sources. All of the examples in Section 1 fit this description. Clilets are *not* suitable for many applications for which one might at first consider them.

The Clilet system is *not* appropriate for applications that use only private data — even if the user does not trust the application. In that case, the user should run the untrusted application in a sandbox that prevents network access. This functionality is available today by running the application in a separate virtual or actual machine. In contrast, some problems require only public data; traditional web applications do just fine for that job. The vast majority of web applications fall into this category.

Again, clilets are best suited for problems that involve the combination of public and private data.

## 7 Future Research

The Clilet system as it stands can implement a number of interesting applications. One direction for future Clilet research involves ways that private data might be selectively shared, in e-commerce for example.

Web-based electronic commerce generally requires users to supply credit card information to merchants. Although they could do so, many merchants do not store credit card numbers because this would make them vulnerable to hackers; they make the users type in the number for every transaction. Security and convenience are unfortunately traded off against each other.

The Clilet system could form an ideal basis for a secure *and* convenient credit card verification scheme that does not require a trusted third party. A user's credit card information would be stored in the secure client-side database. When a transaction is to be made, the server would send a private clilet segment that would read the credit card information and send it back to the merchant. The Clilet system, noting that (private) credit card information is being sent, would verify with the user that this is OK. This semi-manual process is similar to data declassification in a multi-level system.

Since this credit card scheme involves sending private data to the server, it would require that the Clilet system be extended to allow limited data declassification without breaking the security system. The right way to do it for the Clilet system in a general fashion is an open problem.

Clilet ideas might also be extended beyond the domain of web applications. Through the use of HTML, the Clilet system allows a web application to build a user interface with two distinct types of user widgets, public and private. It should be possible to build multi-level user interfaces outside the scope of web applications — a multi-level GUI widget set, for example.

## 8 Conclusion

The web enables more interesting and complex on-line transactions every year. Clilets, which allow web applications to work with private data without the possibility of the disclosure of that data to the server, are another step in this evolution. They are a first step toward allowing the use of private data in a mixed public/private setting.

In the web environment, private data usually originate on the client, while programs originate on the server. Programs and data must find themselves on the same machine in order for any computation to be accomplished: either the data must be sent to the program, or the program must be sent to the data. The Clilet system makes the unusual choice of sending the program to the data — thereby avoiding the risks involved with sending sensitive data to an untrusted server.

In a historical twist, the Clilet system runs untrusted code in the Clilet browser in order to *enhance* user privacy. In the past, untrusted code was seen as a privacy risk. Sandboxing techniques like those used in the Java system — and also by the Clilet system — make it possible to run untrusted code without risking privacy and security.[3] They therefore made feasible the approach taken by the Clilet system — that of moving program to data.

We have shown that the use of private data in web applications is practical by building the Clilet system — a fully functional prototype of the necessary client and server components [3]. Clilet system code and examples can be found at *http://www.clilet.org*.

[3]Sandboxes are only effective to the extent that they are free of bugs that would allow malicious code to break them. The Clilet system assumes the Java sandbox works as advertised; Java sandbox security bugs are outside the scope of this paper.

# References


[1] D. Bell and L. LaPadula. *Secure Computer Systems: Unified Exposition and Multics Interpretation*. The Mitre Corp, 1976.

[2] World Wide Web Consortium. *XHTML 1.1*. `http://www.w3.org/TR/xhtml11/`.

[3] Robert Fischer. Clilet web site. `http://www.eecs.harvard.edu/~citibob/clilet` or `http://www.clilet.org`.

[4] Robert Fischer. "Chapter 7: Clilet HTML — Syntax and Semantics". In *Web Applications with Client-Side Storage, Harvard University Ph.D. Thesis*, pages 113–131, June 2003.

[5] Robert Fischer. "Chapter 8: Browser Implementation". In *Web Applications with Client-Side Storage, Harvard University Ph.D. Thesis*, pages 132–163, June 2003.

[6] Robert Fischer. "Chapter 9: Clilet System Security". In *Web Applications with Client-Side Storage, Harvard University Ph.D. Thesis*, pages 165–187, June 2003.

[7] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 151–160. ACM Press, 1998.

[8] Li Gong. *Inside Java 2 Platform Security*. Addison-Wesley, 1999.

[9] Ira S. Moskowitz and Myong H. Kang. "Covert Channels - Here to Stay?". In *Compass'94: 9th Annual Conference on Computer Assurance*, pages 235–244, Gaithersburg, MD, 1994. National Institute of Standards and Technology.

[10] Andrew C. Myers and Barbara Liskov. "Protecting Privacy Using the Decentralized Label Model". *Software Engineering and Methodology*, 9(4):410–442, 2000.

[11] Andrei Sabelfeld and Andrew Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1), January 2003.

[12] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, Berkeley, California, 2000.

## Appendix A: HTML Semantics

In order to ensure that private clilet segments do not leak data to the server through their HTML, we defined both our own HTML variant (starting with XHTML 1.1) and a semantics that any conforming Clilet browser must follow. The semantics define the kinds of actions taken by the Clilet browser when confronted with each tag. With that information, the HTML Verifier is easily able to determine which tags to prohibit.

Our HTML semantics begin by defining the action of the HTML Display. When presented with a parsed HTML tree, it scans that tree in depth-first search order, building some internal data structures. Those structures are:

1. The *canvas* — an internal representation of the text/graphics page that will be displayed to the user.
2. A set of *forms* found in the HTML page. Every form is associated with exactly one URL.[4]
3. A set of *public form elements* found in the HTML page. There is a many-to-one relation between public form elements and forms. Public form elements are transmitted to the server upon form submission.
4. A set of *private form elements* found in the HTML page. Private form elements act like public form elements, except they are *not* made available to the server upon form submission. Instead, they are made available to the next clilet sent by the server.[5]
5. A set of preloaded external resources, as specified by public `<preload>` tags.

As it scans each node in the parse tree, the HTML Display takes one or more of the following actions:

1. It adds text or graphics to the canvas. For example, an `<h1>` tag or some simple text changes the canvas.
2. It creates a new form; a `<form>` or `<a>` tag, for example. The URL for the form will be sent to the server upon form submission.
3. It adds a public form element to a form; an `<input>` tag in a public HTML segment, for example. Public form elements and their associated values will be sent to the server upon form submission.
4. It adds a private form element to a form; an `<input>` tag in a private HTML segment, for example. Private form elements and their associated values will *not* be sent to the server upon form submission.
5. It initiates a secondary interchange to preload an external resource — a graphic image, for example.
6. It uses a preloaded external resource that was downloaded in a previous secondary interchange.

Our HTML semantics defines the actions taken by the HTML Display upon encountering every kind of tag in Clilet HTML. For example, a `<a>` tag writes to the canvas and creates a form. A `<i>` tag affects only the canvas. A public `<img>` tag affects the canvas and initiates a secondary interchange, whereas a public `<img>` tag with a `ismap` attribute creates a form as well. A private `<img>` tag affects the canvas and initiates a lookup in the set of preloaded URLs; it does *not* initiate a secondary interchange.

The HTML semantics is expressed in tabular form (Appendix B). The semantics of every node in an HTML document is defined by the first row of the semantics table that matches the node. Matching is done based on a node's tag type, its attributes and its public/private designation. Further description of the semantics is available in Fischer03 [4].

### A.1 HTML Verification Rules

The goal of the HTML Verifier is to prohibit all private nodes that *might* result in data being transmitted to the server. Given the above semantics, it is clear that the following must be prohibited in private clilet segments:

[4] Anchors are considered to be forms with zero form elements.

[5] It would seem more intuitive to make private form element values available to the clilet that produced those form elements, rather than to the next clilet. We chose instead to follow the standard practice of HTTP/HTML web applications — in which one servlet creates a form and the next servlet processes it.

1. Nodes that create forms.
2. Nodes that initiate secondary interchanges.
3. Nodes that create public form elements.

Given the HTML semantics, it is easy to determine whether a private node performs one of these actions. The HTML Verifier therefore need only check each private node against a list of criteria. In systematically examining the HTML language, we found the HTML Verifier rules reduce to the following:[6]

1. The `<preload>` tag is added to the Clilet HTML grammar. It is legal only in the public HTML segment.
2. The following tags are not allowed in private HTML segments:

| | | |
|---|---|---|
| `<blockquote cite=...>` | `<q cite=...>` | `<ins cite=...>` |
| `<del cite=...>` | `<a>` | `<link>` |
| `<form>` | `<img ismap=...>` | `<img usemap=...>` |
| `<map>` | `<area>`. | |

3. The following tags create public form elements in a public HTML segment and private form elements in a private HTML segment: `<input>`, `<textarea>`, `<button>`, `<select>`.
4. The following tags cause secondary interchanges in public HTML segments, fetches from the set of preloaded documents in private HTML segments: `<input src=...>`, `<img src=...>`, `<img longdesc=
...>`.

### A.2 Atomicity Rules

The HTML Parser ensures that no node in the parse tree spans a public/private HTML segment boundary. This atomicity requirement is critical to security because it prevents clilets from performing unwanted “tricks” [5]. For example, a public clilet segment might begin an anchor but have a private clilet segment write the destination URL in that anchor tag.

For the same reason, additional atomicity rules are required for some kinds of on-screen HTML elements that require more than one node in an HTML parse tree to describe. For example, an anchor requires two nodes — the `<a>` tags and the text displayed between the begin and end `<a>` tags. Other elements, such as drop-down menus, require even more nodes.

As with other aspects of HTML, atomicity requirements have been systematically and completely examined in our HTML semantics, producing a set of atomicity rules [4]. These rules prevent, for example, a private text node from appearing within a public anchor node. The rules are formulated in terms of “descendants of tag `<X>` must have the same public/private designation as `<X>`.” The set of tags for which this rule holds is precisely: `<a>`, `<area>`, `<map>`, `<button>`, `<optgroup>`, `<select>` and `<textarea>`.

In addition, the `<img usemap=...>` tag can be affected by any other `<map>` node in the document. The HTML Verifier must therefore find the modifier node (if it exists) and ensure that its designation matches that of the original `<img>` node.

[6] Technically, the last two items are the job of the HTML Display, not the HTML Verifier

# Appendix B: HTML Semantics Table

| Public/Private C = public R=private | Node Type | Attribute | Canvas | Forms | Public Form Elements | Private Form Elements | 2º Interchange | Preload Lookup | Modified By |
|---|---|---|---|---|---|---|---|---|---|
| | ***Blocks*** | | | | | | | | |
| | acronym | | x | | | | | | |
| | abbr | | x | | | | | | |
| | address | | x | | | | | | |
| C | blockquote | cite | x | x | | | | | |
| | blockquote | | x | | | | | | |
| C | q | cite | x | x | | | | | |
| | q | | x | | | | | | |
| | cite | | x | | | | | | |
| C | ins | cite | x | x | | | | | |
| | ins | | x | | | | | | |
| C | del | cite | x | x | | | | | |
| | del | | x | | | | | | |
| | ***Links*** | | | | | | | | |
| C | a | | x | x | | | | | children |
| C | link | | x | | | | x | | |
| R | link | | x | | | | | x | |
| | ***Input*** | | | | | | | | |
| C | form | | | x | | | | | |
| C | input | src | x | | x | | x | | |
| R | input | src | x | | x | | | x | |
| C | input | | x | | x | | | | |
| R | input | | x | | | x | | | |
| C | textarea | | x | | x | | | | children |
| R | textarea | | x | | | x | | | children |
| C | button | | x | | x | | | | children |
| R | button | | x | | | x | | | children |
| C | select | | x | | x | | | | children |
| R | select | | x | | | x | | | children |
| | optgroup | | x | | | | | | children |
| | option | | x | | | | | | |
| | label | | x | | | | | | |
| | fieldset | | x | | | | | | |
| | legend | | x | | | | | | |
| | ***Images*** | | | | | | | | |
| C | img | ismap | x | x | | | x | | |
| C | img | usemap | x | x | | | x | | map |
| C | img | src,longdesc | x | | | | x | | |
| R | img | src,longdesc | x | | | | | x | |
| C | map | | | x | | | | | children |
| C | area | | | x | | | | | |
| | ***Meta Info*** | | | | | | | | |
| | head | | x | | | | | | |
| | title | | x | | | | | | |
| | meta | | x | | | | | | |
| C | base | | x | | | | x | | |

| | |
|---|---|
| ***Programming*** | No client-side execution (other than clilets) in Clilet HTML<br>Following tags are removed: *script, noscript, applet, object, param* |
| ***Frames*** | No Frames in Clilet HTML<br>Following tags are removed: *frame, frameset, noframeset, iframe* |
| ***Styles*** | Style tags affect canvas only<br>Tags are: *style, div, span* |
| ***Basic Tags*** | Basic tags affect canvas only<br>Tags are: *html, body, h1, h2, h3, h4, h5, h6, p, br, hr* |
| ***Char Format*** | Formatting tags affect canvas only<br>Tags are: *b, font, i, em, big, strong, small, sup, sub, bdo, u* |
| ***Tables*** | Table tags affect canvas only<br>Tags are: *table, caption, th, tr, td, thead, tbody, tfoot, col, colgroup* |
| ***Output*** | Output tags affect canvas only<br>Tags are: *pre, code, tt, kbd, var, samp* |
| ***Lists*** | List tags affect canvas only<br>Tags are: *ul, ol, li, dir, dl, dt, dfn, dd, menu* |

The table on the preceding two pages describes Clilet HTML Semantics. The semantics of every node in an HTML document is defined by first row of the semantics table that matches the node. Matching is done based on a node's tag type, its attributes and its public/private designation.

The columns at the right describe the action taken by the HTML display upon encountering a matching tag. The *Modified By* column indicate additional atomicity requirements (Section 8). Most nodes are modified by their children; the `<img usemap=...>` node is potentially modified by any `<map>` node in the document.