Visual Security Policy for the Web

Terri Oda and Anil Somayaji
Carleton Computer Security Laboratory
Ottawa, Ontario, Canada
{terri, soma}@ccsl.carleton.ca

Abstract

Many web security vulnerabilities allow parts of a page to interact when they should be isolated. Such vulnerabilities can be mitigated by implementing protection boundaries between web page elements. Several methods exist for creating such boundaries, but existing methods require relatively sophisticated knowledge of web technologies. To make protection mechanisms available to a wider audience, we propose a simple web page security policy language, ViSP, modelled on mechanisms for specifying page layout. Here we characterise ViSP and describe a simple Firefox-based prototype that allows interactive, graphical specification of per-page security policies. We also show how these tools can be used to protect against cross-site scripting (XSS) attacks on common web applications.

1 Introduction

The web is currently the largest exploit vector in computer security: Over 80% of computer vulnerabilities involve the web [3]. Many web-based exploits make use of JavaScript's ability to do intra-page modifications; as a result, one popular strategy for mitigating vulnerabilities is to isolate portions of a web page from each other. Many researchers have recognized the security implications of this situation and have developed encapsulation mechanisms for web mashups [8, 16, 5, 1]. Given the potential vulnerabilities in common included content such as advertisements, search boxes, and embedded video [13], it is arguable that many, perhaps even *most* web pages should be using sub-page isolation mechanisms to mitigate XSS vulnerabilities. Existing approaches, however, require a significant amount of programming expertise to be used effectively—something not possessed by the artistic professionals and amateurs that create many websites [12].

What is needed, then, are isolation mechanisms that can be understood and used by all web page creators. These mechanisms cannot require knowledge of programming language conventions; instead, they need to be in a language they already understand. Here we argue that the language of page layout provides the necessary vocabulary for specifying protection boundaries within web pages. Layout-based protection boundaries can be specified and enforced in terms of the bounding boxes used to visually separate web page elements. Layout-based policies can be specified without reference to the code underlying a page. They can be stored separately, much as a style sheet. And, they can be updated and manipulated using tools that are no more complex than those used to create and maintain web pages, all while providing protection against many forms of cross-site scripting attacks, the most pervasive category of security vulnerabilities on the Internet today [4].

To support these claims, we have developed ViSP (Visual Security Policy), a simple XML-based policy language for specifying protection boundaries and allowed interactions that is modelled on standard ways of specifying web page layout. Because ViSP directly maps to page layout conventions, developers can quickly specify which page elements should be isolated and which should be allowed to interact simply by graphically selecting those elements and selecting their policies. ViSP then can be enforced by rewriting a page to use other, standard isolation mechanisms; alternately, web browsers can be modified to enforce ViSP directly.

To show the utility of ViSP, we explain how a simple ViSP policy could mitigate an intra-page XSS attack by an advertiser on a site. (Note this attack scenario is not addressed by many existing XSS mitigation mechanisms because they must whitelist ad servers.) We also present a remarkably simple but effective policy for Facebook, a (very) popular social networking site that has been shown to have a number of XSS vulnerabilities in the past. For future work we plan to test ViSP on a wider variety of websites and to evaluate ViSP's usability through lab studies with amateur web developers.

The rest of this paper proceeds as follows. §2 describes the design of the ViSP policy language. §3 presents the details of ViSP's syntax. §4 describes a simple attack and how ViSP can mitigate it; §5 presents a ViSP policy for Facebook. §6 presents our prototype implementation of ViSP and discusses alternative implementation strategies. §7 discusses our experiences with ViSP, its limitations, and our plans for future work. §8 concludes.

2 Visual Security Policy (ViSP)

Visual Security Policy (ViSP) is an XML-based security policy language whose construction is based upon the layout of the visual elements of a page. ViSP provides a way of specifying compartmentalization of an HTML page in terms of drawing visual boxes on the layout. Existing web mashups work has concentrated on the HTML representation of the page rather than the layout and achieved good results. It is our hope that concentrating on the layout will yield usability improvements.

There are a few reasons to believe this may be true. First, the visual representation of the page is accessible to more people. While many people have little to no understanding of the underlying HTML, they are typically familiar with the appearance and behaviours of a page.

Second, designers are used to separating the page content from the page layout and style: the content is contained within HTML and the style is largely contained within the associated Cascading Style Sheets (CSS). The separation between content and style is often cited as something which will make long-term maintenance of the page easier. Providing security as something akin to another type of stylesheet may yield similar benefits.

It may seem initially that layout and security have little to do with one another. However the appearance of a page is highly correlated with the way in which it is intended to be used and understood. Usability of web pages is closely linked to both convention and design reuse. Steve Krug states that his first law of [web] usability is, “Don’t make me think” [10]. The idea is that to be usable, pages should be as self-evident as possible. There are a variety of ways to make the page more self-evident, but many of them involve using familiar buttons, navigation, and other design patterns. The idea of obviousness equating with usability is borne up by user studies conducted by Nielsen et al. [11], which suggest that non-standard controls are among the worst flaws that can be found in web design.

Usability is centred around encouraging the user to behave in ways that the system expects; security is about forcing users of the system to behave in correct ways. So if the layout of a page is already designed to help a user behave in a given way, using the same layout to add security requirements to the page allows people to more easily extend existing models to require secure behaviours.

One final benefit to using the visual parts of a page to define security is that doing so makes the security of a page much less likely to be in conflict with its appearance. This is in some ways the opposite to the previous comment: while design shapes the use of the page and can be used to help shape security, the opposite is also true in that security concerns can result in design modifications. Working visually allows the system to help provide some shortcuts to make it easier to go from a pre-existing design to a more secure version of the same design. As programmers, it may be tempting to ignore minor design flaws as irrelevant. But given the existing barriers towards improving web security, and given that the appearance of the page can directly affect usability, it can be surprisingly important to get appearances just right.

3 The ViSP Language

While the idea of ViSP is that policies can be represented visually, for programmatic evaluation and manipulation, it is useful to also have an underlying textual representation of the policy. As such, ViSP is a simple, XML-based language inspired by standard HTML layout.

A visual policy only needs to refer to the larger, visible regions within a page. HTML already has a tag for referring to such regions, the `<div>` tag. In our initial experiments we attempted to use simplification of the page which included only the HTML `<div>` tags. Unfortunately, this proved to be insufficiently robust since it relied upon the page being designed to use `<div>` tags and made it impossible to apply policy to some smaller regions. This also didn’t allow us a clear separation between policy and the page itself.

To address these problems, the ViSP language uses tags analogous to, but different from standard HTML tags. The focus of the ViSP language is to only describe the regions that are of interest security-wise, the necessary structure to explain the visual layout of these regions, and the basic communications channels between them. We also wanted to make it easy to describe regions with multiple pieces of user-generated content that all should be separated from each other. These design goals resulted in four tags from which a basic visual policy can be constructed as a simplification of the original HTML page. Figure 1 gives a quick visual overview of ViSP. The four tags are as follows:

box The box tag defines a region of interest within the HTML, one for which we wish to set security properties and possibly communications channels. These are shown using solid boxes.

structure The structure tag defines layout which does not have security properties of its own but which is necessary to give the layout of defined boxes. These are not shown on the diagrams.

channel The channel tag, placed within a box, defines a single communication channel from another box to the box where it is defined. This enables creation of a directed graph of communications channels. Note that the communications channels are not symmetric: the menu of a page might be allowed to change the content, while the content is unable to modify the menu. These are shown using a black arrow.

multibox The multibox tag is a shortcut for a common construct within HTML pages. Rather than being a box itself, the multibox indicates that all sub-boxes of this HTML element should be listed as separate

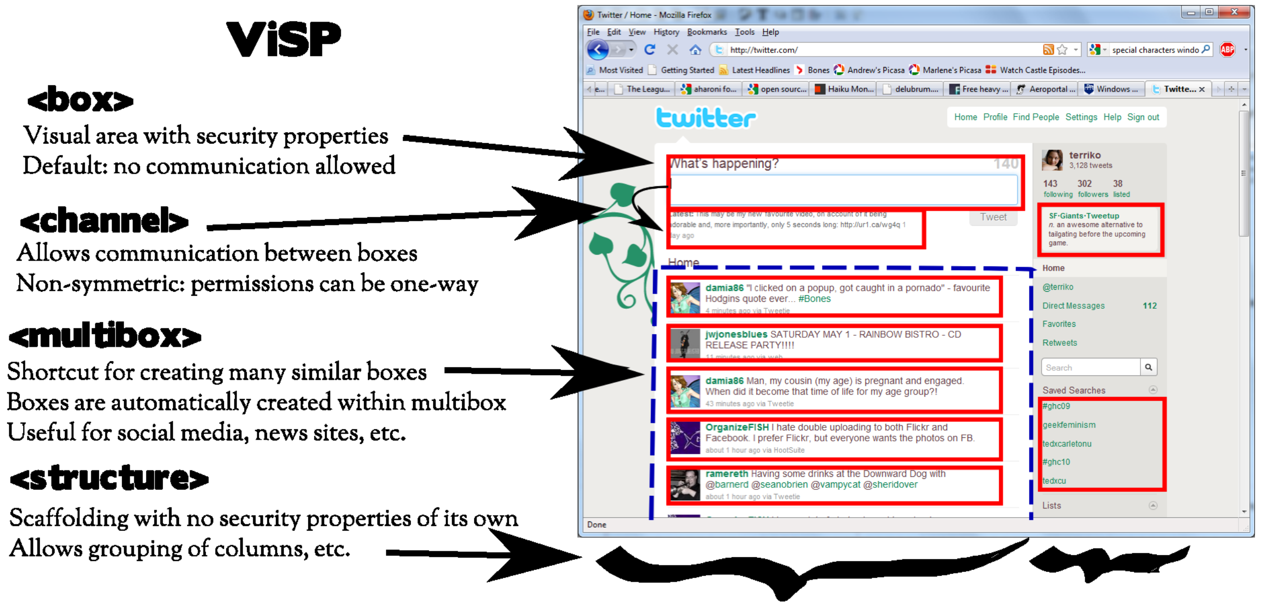

ViSP
<box>
Visual area with security properties
Default: no communication allowed
<channel>
Allows communication between boxes
Non-symmetric: permissions can be one-way
<multibox>
Shortcut for creating many similar boxes
Boxes are automatically created within multibox
Useful for social media, news sites, etc.
<structure>
Scaffolding with no security properties of its own
Allows grouping of columns, etc.
twitter
What's happening?
Home

Figure 1: Overview of ViSP

boxes. These are shown using dashed boxes, and the sub boxes generated from a multibox will be shown as solid boxes. The boxes created within a multibox are by default fully isolated, just like any other newly-created box.

4 A Simple Attack

To demonstrate the use of visual policies, consider an example based upon a real site and a hypothetical exploit. CNET provides reviews for a variety of consumer electronics, including phones. Like many other companies, CNET runs advertisements on sites that review their products. This is a good place for targeted advertisements, as those looking at reviews are often planning on buying a similar product. Figure 2 shows advertisements on CNET's review section. The review is for the Palm Pre, and one of the advertisements being displayed is for a competing smartphone, the Blackberry Curve.

On a review site, like in a traditional print magazine, the advertisements are separated from the review text using layout cues and text such as "paid advertising section." While such cues distinguish advertisements from text visually, advertisements on a web page may include JavaScript code that could change other parts of the page, including the contents of a competitors review. Although there is no evidence of wrongdoing on the part of the companies displayed in this example, it is not unheard for for companies to use underhanded tactics to improve their reviews [14].

For this example, suppose that a malicious advertiser wishes to alter the final rating given to the phone. Sample JavaScript which could do this is shown in Listing A.

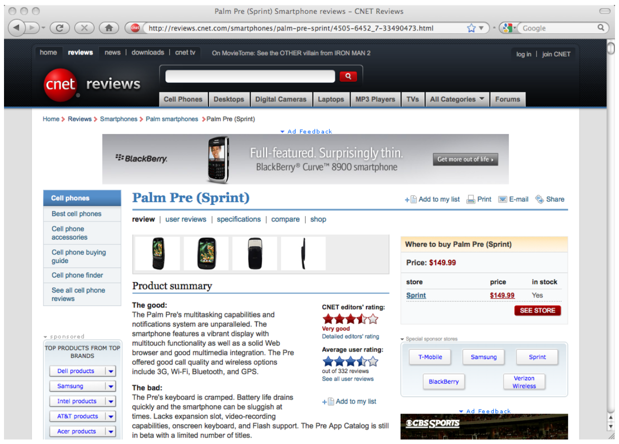

cnet reviews
Palm Pre (Sprint)
Full-featured. Surprisingly thin.
Product summary
Where to buy Palm Pre (Sprint)

Figure 2: Original CNET page.

Listing A: JavaScript code used to change the CNET rating to a 1 or Very Poor rating.

```
// grab the rating section
edStars = document.getElementById("edStars");

// Find the span with the numerical rating
// and change it
spans = edStars.getElementsByTagName("span");
for (i = 0; i < spans.length; ++i) {
    if (spans[i].className = "rating") {
        spans[i].innerHTML = 1.0;
    }
}

// update the interior text
edStars.innerHTML = edStars.innerHTML.replace(
    /Very Good/ig, "Very Poor");

// update the actual stars display CSS
links = edStars.getElementsByTagName("a");
links[0].className = "edRate1 toolTipElement";
```

To block this attack, advertisements must be isolated from

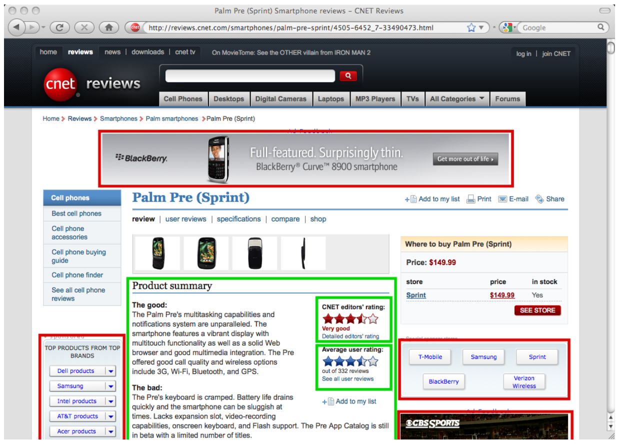


Figure 3: CNET page with visual policy.

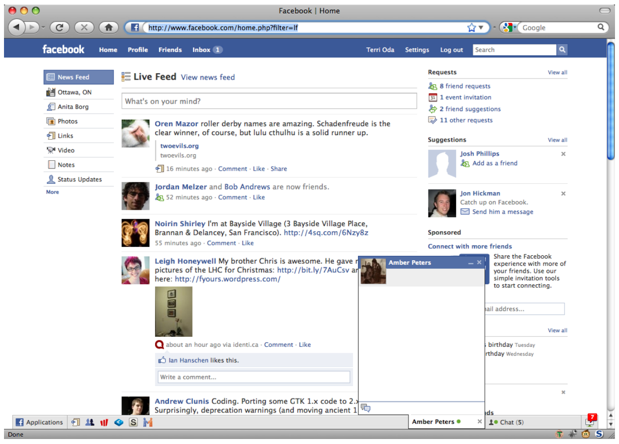


Figure 4: Homepage for a logged-in Facebook user

the review content. They are visually distinct, but we need to compartmentalize them to match the page's layout.

Figure 3 gives a simple sample policy that does exactly that. The advertising features are enclosed in boxes which are red, and the review parts of the page are enclosed in green boxes. This colouring is just for the purpose of discussing the boxes—there need not be any functional difference in the encapsulation. The corresponding XML version of this same policy is given in Listing B.

Listing B: XML Visual Policy for CNET Review

```
<structure alt="Whole page">
  <box id="div:1" alt="Ad Banner" />
  <structure alt="Columns">
    <box id="div:2" alt="Sponsored left" />
    <structure alt="Column 2">
      <box id="div:contentBody" alt="Review">
        <box id="div:edStars" alt="Editor *s"/>
        <box id="div:userStars" alt="User *s"/>
      </box>
    </structure>
    <structure alt="Column 3">
        <box id="div:3" alt="Sponsored right" />
        <box id="div:4" alt="Advertising box" />
    </structure>
  </structure>
</structure>
```

For the purposes of this example, assume that the policy setting for each box allows absolutely no communication in or out. Given that there is no need for the advertisements to modify the review, and plenty of reasons that it would be inappropriate for them to do so, this is a reasonable policy setting. (Although it is worth noting that the advertisement server may prefer to have at least read access to the content of the page to better target advertisements, let us assume a more conservative policy for the sake of simplicity.)

The attack code, as shown in Listing A, needed to gain access to the tag with the id "edStars." However, in Figure 3 the review stars are contained within a visual policy box, meaning they are protected from other parts of the page. Similarly, the advertisement where the attack code is concealed has its own box, so the attack code is cut off from all of the page, not just the parts which have their own visual policy boxes. Thus, the attack will fail: the advertisement can modify only its own banner.

Note that common mitigation strategies such as tainting whitelist advertisement servers [15, 6, 9]; as a result, they cannot defend against this attack.

5 ViSP for Facebook

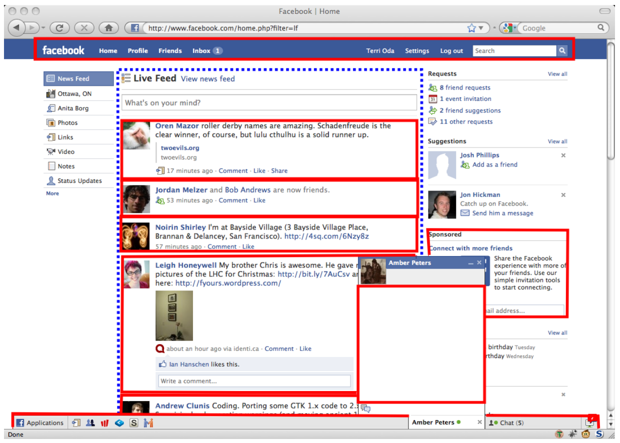
In the US, Facebook now accounts for 25% of total page views on the Internet [7]. It undeniably has a huge impact upon the web, and it is important that any web security solution be able to deal with Facebook or pages based upon the popular look and feel of the site. Figure 4 shows the home page of a logged in user on Facebook [1].

The page is very busy, including status updates, a chat box (or chat boxes if you are talking to multiple users), a sponsored advertisement on the right hand side, menus at top, bottom and sides of the page, and a variety of other information displayed. At first glance, it may appear daunting. However, thanks to the multibox structure, we can easily group the centre column's status messages rather than having to manually set policy for hundreds of status updates. We might additionally be able to do this with the left and right columns for some pages. As such, ViSP for this part of Facebook can be something like what is shown in Figure 5, with the corresponding XML given in Listing C.

Listing C: ViSP XML for Facebook home page

```
<box id="div:fb_menubar" alt="Top menu" />
<structure>
  <multibox id="div:home_stream"
     alt="Status updates"
     boxspec="div:class:GenericStory" />
  <box id="div:83" alt="Sponsored box" />
</structure>
<box id="div:presence_bar" alt="bottom menu">
  <box id="div:chat_conv"
```

[1]This does not reflect the most recent design. Facebook changes their interface regularly but many redesigns share similar elements.



Oren Mazor roller derby names are amazing. Schadenfreude is the clear winner, of course, but lulu cthulhu is a solid runner up.
twoevils.org
twoevils.org
17 minutes ago · Comment · Like · Share

Jordan Melzer and Bob Andrews are now friends.
53 minutes ago · Comment · Like

Noirin Shirley I'm at Bayside Village (3 Bayside Village Place, Brannan & Delancey, San Francisco). http://4sq.com/6Nzy8z
57 minutes ago · Comment · Like

Leigh Honeywell My brother Chris is awesome. He gave pictures of the LHC for Christmas: http://bit.ly/7AuCsv ar here: http://fyours.wordpress.com/
about an hour ago via identi.ca · Comment · Like
Ian Hanschen likes this.
Write a comment...

Andrew Clunis Coding. Porting some GTK 1.x code to 2.

Share the Facebook experience with more of your friends. Use our simple invitation tools to start connecting.

Figure 5: ViSP for Facebook

```
        alt="Chat  conversation"  />
</box>
```

This is not the only possible ViSP for Facebook – one might want to add additional protections for other menus or content displayed in the left and right columns, or one might want to relax some of these restrictions, depending upon Facebook's own goals and those of its users. However, the example shows that even with a fairly complex site, the policy can be surprisingly small and manageable.

6 Implementing ViSP

A ViSP policy creation tool has been implemented in JavaScript as a Firefox 3 add-on. Once installed, it adds a menu option allowing the user to enter a policy-creation mode. In this mode, moving the mouse over the page highlights page elements, one at a time, when the mouse is over them. The current tool does so by showing a yellow border around the page element. The user then mouses over the desired page element and clicks to add it to the visual security policy. Once added to the policy, the border around that element becomes red and permanent, staying even when the mouse exits the area.

The other necessary ViSP tool is one which will handle enforcement of policies. But at what level should we translate and enforce the policy? There are several possible locations. The web developer might take the ViSP policy for the page and use some tool to create a new page which includes the compartmentalization described within the policy. Similarly, a script on the web server or on a proxy server could translate the pages before they are delivered to the user. Finally, the user's web browser itself might be the final arbiter of any ViSP data. This method has the advantage that more appearance data can be used, but the disadvantage that it requires modifications to browsers while the others can use current technologies.

The prototype ViSP policy enforcement tool currently takes as input the page and the policy, and produces a new page which uses `iframes` to provide basic encapsulation. The script used for enforcement could be used by the web developer, or put inline on the web server or a proxy so that it can be used directly on existing web applications that use more dynamic code. The use of iframes currently results in some minor irregularities, but it is our hope that future versions can be more faithful renditions of the original page. Full implementation of ViSP, however, will likely require deep browser integration as ViSP is not lexically scoped—enforcement engines must take into account the non-local interactions of HTML, CSS, and JavaScript elements.

7 Discussion

It is important to note that ViSP has a number of limitations, even within the focus of isolating regions of a web page from each other. ViSP has no support for isolating code or data that are not visually represented, e.g., code in page headers. It cannot specify partial access between regions, say by originating domain or content type. Also, because our current prototype enforcement engine uses standard `iframe` tags, it produces clear visual artifacts. It may be easier to fix this problem when we can use new language constructs in HTML5 such as their seamless `<sandbox>` attribute [2] .

We created basic ViSP policies for 15 web sites, specifically targeting blogs and other smaller sites that are often run by amateurs for their own personal interest. We examined a larger number of sites (200) to determine whether they were likely to follow similar patterns and determine the viability of ViSP before the language was fully set.

One surprising finding is that surprisingly few of the pages we examined required communication channels of any sort. Many pages use cut-and-paste code inserts: advertisements, Twitter feeds, Flickr badges, etc. that are designed so that they can be inserted anywhere. These can be isolated without incident. What is perhaps more surprising is that menus and media inserts followed similar patterns. Although there is no reason for code to be inserted only near where it is used, the reality is that common programming style choices result in easily-encapsulated code. There were a few exceptions where top-level JavaScript needed access to boxes within the page (such as for advertisements), but for the most part the pages could be divided up with little communication necessary between page elements.

This tendency towards easy encapsulation may be a side effect of choosing sites which are likely to be created by amateurs. Perhaps it is not too surprising that these sites use only a smaller, simpler subset of the capabilities of the web. This suggests that ViSP is indeed viable for these smaller, amateur sites which it is designed to protect. It is less clear at this stage as to whether ViSP can be helpful with more complex sites, and whether complex sites are more rare than one might expect.

We have chosen to trade off expressiveness for simplicity in order to produce a policy language that directly maps to visually representable boundaries. This is in the hope that such simplicity will make ViSP easier to learn and use regardless of background. We are currently in the process of setting up user tests to validate this hypothesis; informal user testing has already yielded positive results.

Although here we created a separate language for ViSP in order to reduce pages to a set of security-relevant boxes, future work may focus on using existing CSS and HTML constructs more directly. This would allow page creators to produce a "security stylesheet" using CSS syntax to specify security constraints like visual styles. The hope is that this will be easier to integrate into existing tools and take advantage of existing knowledge, as well as make it possible to integrate such a security system into HTML5.

We do not see ViSP as a replacement for other approaches to implementing protection boundaries inside web pages; rather, we see ViSP as an approach that is "good enough" for many web pages. More importantly, we believe something like ViSP is necessary for most web pages in order to limit the impact of XSS attacks and malicious third-party inclusions. ViSP, then, is a small step towards security mechanisms that regular web designers will use because they give them clear benefits while imposing few costs.

8 Conclusions

While methods already exist to create more security-hardened pages using protection boundaries, these methods are designed with programmers in mind and often require a significant amount of learning, effort and time to implement on an existing website. As such, it is hard for these otherwise good solutions to gain traction among web page creators and maintainers who may not have programming skills or the time necessary to learn and implement security enhancements. ViSP deals with this problem by providing a simpler method for creating web security policies, one which is based upon the visual layout of a page. Because it is based in the visual realm in which many designers and users think, it is much easier to understand at a glance and is easier to specify, all while still protecting against a wide variety of XSS and malicious inclusion attacks. ViSP is thus intended to be a solution which better meets the needs of those who create and maintain web pages.

Acknowledgements: This work was supported by NSERC through ISSNet (www.issnet.ca) and the Discovery grant program. We also thank CCSL members who have provided ongoing feedback for this research.

References

[1] google-caja: A source-to-source translator for securing javascript-based web content, 2009. `http://code.google.com/p/google-caja/`.

[2] HTML 5: A vocabulary and associated APIs for HTML and XHTML. Technical Report 1.2852, World Wide Web Consortium (W3C), Aug 2009. `http://www.w3.org/TR/2009/WD-html5-20090825/`.

[3] Web application security trends report – Q3-Q4, 2009. *Cenzic Inc.*, 2009.

[4] S. Christey and R. A. Martin. Vulnerability type distributions in CVE. Technical Report 1.1, MITRE Corporation, May 22 2007.

[5] F. De Keukelaere, S. Bhola, M. Steiner, S. Chari, and S. Yoshihama. Smash: Secure cross-domain mashups on unmodified browsers. Technical report, IBM Research, Tokyo Research Laboratory, 2007.

[6] M. Dhawan and V. Ganapathy. Analyzing information flow in javascript-based browser extensions. In *Proc. of the 25th Annual Computer Security Applications Conference (ACSAC'09)*, Honolulu, Hawaii, Dec 2009.

[7] B. Heater. Facebook accounts for 25 percent of page views. *PCMag*, 2009.

[8] J. Howell, C. Jackson, H. J. Wang, and X. Fan. MashupOS: Operating system abstractions for client mashups. In *Workshop on Hot Topics in Operating Systems*, 2007.

[9] InformAction. Noscript. `http://noscript.net/`.

[10] S. Krug. *Don't Make Me Think: A Common Sense Approach to Web Usability*. New Riders Press, 2nd edition, Aug 2005.

[11] J. Nielsen and H. Loranger. *Prioritizing Web Usability*. New Riders Press, Apr 2006.

[12] T. Oda and A. Somayaji. No web site left behind: Are we making web security only for the elite? In *Web 2.0 Security and Privacy (W2SP)*, May 20 2010.

[13] T. Oda, A. Somayaji, and T. White. Content provider conflict on the modern web. *Symposium on Information Assurance (New York State Cyber Security Conference)*, 2008.

[14] A. Parsa. Fresh evidence suggests belkin's amazon sales rep was engaged in more unethical activities. *The Daily Background*, Jan 2009. `http://www.thedailybackground.com/2009/01/19/`.

[15] P. Vogt, F. Nentwich, N. Jovanovic, C. Kruegel, E. Kirda, and G. Vigna. Cross site scripting prevention with dynamic data tainting and static analysis. In *14th Annual Network and Distributed System Security Symposium (NDSS 2007)*, San Diego, CA, Feb 2007.

[16] H. J. Wang, X. Fan, J. Howell, and C. Jackson. Protection and communication abstractions for web browsers in MashupOS. In *21st ACM Symposium on Operating Systems Principles (SOSP)*, 2007.