SpyShield: Preserving Privacy from Spy Add-ons

Zhuowei Li, XiaoFeng Wang and Jong Youl Choi

School of Informatics, Indiana University at Bloomington, USA
email:{zholi, xw7, jychoi}@indiana.edu

Abstract. Spyware infections are becoming extremely pervasive, posing a grave threat to Internet users' privacy. Control of such an epidemic is increasingly difficult for the existing defense mechanisms, which in many cases rely on detection alone. In this paper, we propose SpyShield, a new *containment* technique, to add another layer of defense against spyware. Our technique can automatically block the visions of untrusted programs in the presence of sensitive information, which preserves users' privacy even after spyware has managed to evade detection. It also enables users to avoid the risks of using free software which could be bundled with surveillance code. As a first step, our design of SpyShield offers general protection against spy add-ons, an important type of spyware. This is achieved through enforcing a set of security policies to the channels an add-on can use to monitor its host application, such as COM interfaces and shared memory, so as to block unauthorized leakage of sensitive information. We prototyped SpyShield under Windows XP to protect Internet Explorer and also evaluated it using real plug-ins. Our experimental study shows that the technique can effectively disrupt spyware surveillance in accordance with security policies and introduce only a small overhead.

1 Introduction

Spyware is rapidly becoming one of the most dangerous threats to the nation's critical information infrastructure. Webroot estimated that about 89 percent of consumer computers are riddled with spyware in this country with an average of 30 pieces per machine [4]. A recent study [19] further shows that a large portion of spyware infections are in the form of add-ons to common software such as Internet Explorer (IE). These add-ons seriously threaten the safety of personal identity information, as they can be used to stealthily collect from users sensitive data such as passwords, credit card numbers and social security numbers.

Add-ons are optional software modules which complement or enhance a software application they are attached to (called a *host application* or simply a *host*). Examples of these modules include Microsoft's plug-ins [1] and Mozilla's extensions [3]. Software manufacturers usually offer standard interfaces for third parties to develop their own add-ons, which we call *add-on interfaces*. Through such interfaces, a spy add-on may acquire sensitive information from the host application or even control it.

The threat posed by spy add-ons is recognized as an important security concern and has recently received great research attentions [19, 15]. Existing defense against such spyware heavily relies on detection techniques. Specifically, spyware scanners are used to search binary executables for the presence of binary-pattern signatures which appear in a spyware database. Signature-based detection can be evaded by metamorphic and polymorphic spyware which transforms its code for every new infection. An alternative is behavior-based detection [19] which employs dynamic analysis or static analysis to capture spyware's surveillance activities. Although this technique is more resilient to metamorphism, it could still be got around by the spyware which exhibits unconventional behaviors, for example, direct reading of sensitive data from process memory.

Since no detection techniques are absolutely reliable, we have to consider an in-depth defense strategy: in case a piece of spyware penetrates other layers of defense, protection must still be there to save important information from being stolen. In addition, since surveillance code could be bundled with useful and often free software, it becomes highly desired to enable users to use such software while avoiding the potential risk it brings about. Serving these purposes is the technique of *spyware containment*, which strives to preserve clients' privacy in the presence of malicious surveillance. Existing research on this subject is limited to the techniques which provide a trusted input path for passwords [21, 17]. These techniques are inadequate to contain spy add-ons which can also snoop on other important data, for example, the account balance displayed in a browser.

In this paper, we present the first spyware-containment technique which offers general protection against the surveillance from spy add-ons. Our approach, called *SpyShield*, can automatically block the view of an untrusted add-on whenever sensitive data are being accessed by its host application. This is achieved through a proxy which enforces security policies to add-on interfaces. For example, our approach ensures that whenever an IE browser is visiting `citi.com`, no data can flow through a COM interface into an untrusted plug-in. While it is impossible to get the privacy via COM interfaces, spy add-ons could bypass the proxy through direct memory access. SpyShield addresses the concerns by separating untrusted add-ons from their host's process.

We prototyped SpyShield on Windows XP and evaluated it using known spyware. Our implementation effectively blocked their surveillance attempts in accordance with a set of security policies. We also demonstrate that our technique introduces small performance overheads. We believe that SpyShield advances the state-of-the-art of spyware defense in following perspectives:

- **General protection against spy add-ons**. SpyShield offers the first general avenue to protect sensitive information from untrusted add-ons. Our design works for different add-on interfaces, such as COM and XPCOM [7], and therefore can be used in the applications adopting these interfaces, such as Internet Explorer, Microsoft Outlook, Mozilla Firefox.
- **Fine-grained access control**. We propose a new policy model, called *sensitive zone*. An application enters a sensitive zone whenever it starts processing sensitive data. Inside that zone, our approach allows defining and enforcing fine-grained access policies. For example, we may grant untrusted plug-ins free access to unimportant data on a web page such as advertisements, but forbid them to read and write sensitive data such as passwords.
- **Resilience to attacks**. SpyShield can protect itself from being attacked. It utilizes a lightweight kernel driver to prevent unauthorized modification of the proxy's code

and data, and any attempts to load untrusted code into the kernel of an operating system (OS).

- **Small overheads**. Our research further shows that the overhead of SpyShield, which is mainly caused by cross-process communications, may not be significant enough to be perceived by the user, as it could be overshadowed by the delay for accomplishing an add-on's normal mission.
- **Ease of use**. SpyShield does not require modifying host applications and OS settings. Users do not need to change their behaviors when using it, though they can choose to modify default security polices through a secure and user-friendly interface. SpyShield can also be easily turned off and on.

The rest of the paper is organized as follows. Section 2 presents the design of SpyShield. Section 3 describes our implementation of a prototype system. Section 4 reports the evaluations of our technique. Section 5 discusses its limitations. Section 6 reviews the related approaches and compares them with SpyShield. Section 7 concludes the paper and envisions the future research.

2 Design

SpyShield inserts an access-control proxy between untrusted add-ons and their host application to control their communications according to a set of security policies. Based on the method how to interpose communications, SpyShield can be implemented in two ways: either one-process or two-process solution. While in one-process solution add-ons and the host application coexist inside a same process, SpyShield can separate them into two different processes so that we can put a process barrier to inhibit untrusted add-ons from accessing the memory space of the host application to obtain any sensitive information. Figure 1 illustrates an example using Internet Explorer (IE)

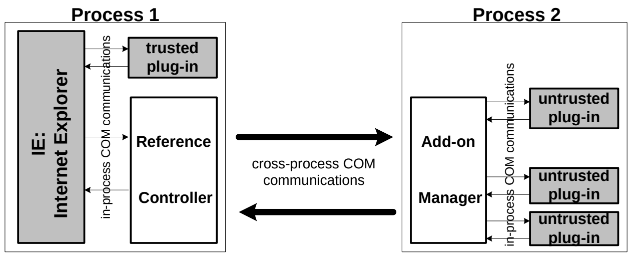


Fig. 1. A containment mechanism for untrusted plug-ins (e.g., BHO, toolbar) in Internet Explorer.

as the host application. The proxy in the Figure consists of two components, a *reference controller* in the form of an IE plug-in, and an *add-on manager* serving as an independent process which handles a set of untrusted plug-ins. To these plug-ins, the add-on manager plays the role of an IE browser, which automatically loads them into memory and offers standard COM interfaces to enable them to subscribe to events and ask for information of their interest. Actual invocation of COM interface [31], however, is delegated to the reference controller by transporting add-ons' requests through a cross-process communication channel. Upon receiving each request from plug-ins, the reference controller will make a decision regarding whether to forward the request to the browser. The decisions will be based on a set of security policies pre-defined by a user. IE's event or responses should go through the security policy enforced by the reference controller. With this approach, we can prevent a spy plug-in from stealing information through either the COM interfaces or direct access to the browser's memory. An end user, on the other hand, will have more controls of her information by adjusting security policies.

To defeat any attempts from thwarting the access-control proxy, the proxy can be overseen by a kernel driver, called *proxy guardian*, which prevents unauthorized attempts to temper with the proxy's data and code. Although we use IE as an example here, the architecture is general enough to work on other add-on interfaces such as XPCOM [7] and other applications such as Mozilla Firefox.

2.1 Access-control Proxy

The objective of the access-control proxy is to permit or deny add-ons' access to their host application's data based on security policies. This is achieved through collaborations between the reference controller in the form of an application's add-on, and the add-on manager which hosts untrusted add-ons. After an untrusted add-on is loaded, its request to subscribe to an event is intercepted and recorded by the add-on manager which informs the reference controller to register that event using an event-handling function (called a *callback* function). The occurrence of the event first triggers that function which then decides whether to invoke the add-on and pass to it the parameters received from the application.

Though most spyware add-ons are event-driven, there are exceptions: for example, `UCMore` [9] toolbar can poll the COM interfaces of an IE browser for the URLs and the web pages visited recently. To contain such spyware, an access-control proxy needs to interpose on all add-on interfaces. In the above example, the add-on manager can implement `IWebBrowser2`, a COM interface which offers add-ons methods such as `get_LocationURL` and `get_Document` for accessing URLs and web pages. This allows the reference controller to block all undesired invocations of these methods.

An add-on may attempt to directly interact with its host application, without going through an add-on interface. For example, a Windows toolbar may requests from a COM interface a handle of a browser's window for directly retrieving its content. In this case, the add-on manager needs to create that window's substitute for the toolbar and selectively copy data to it according to security policies.

An important design issue is the choice between the solution which keeps a host application, the proxy and add-ons inside the same process, and the alternative which separates the add-on manager and untrusted add-ons from the host and the reference controller. The one-process solution gives good performance, which avoids expensive cross-process communications. However, it leaves the door open for the attacks using direct memory access. The two-process solution separates the untrusted add-ons from their host application's process, and therefore eliminates the threat originated from direct memory access. This approach also protects a host application from the add-ons containing security flaws which may crash the application or be exploited by attackers. Its weakness is performance, which suffers from cross-process communications

Name	Policy	Comments
Browser rule	`IDispatch`$\longrightarrow$`Invoke(Event)` $\longmapsto$`decline`	Block an IE browser's attempt to trigger untrusted plug-ins through calling the `invoke` functions of their `IDispatch` interfaces.
Plug-in rule	`IWebBrowser2`$\longrightarrow$`get_LocationURL`| `IWebBrowser2`$\longrightarrow$`get_Document`| `IWebBrowser2`$\longrightarrow$`navigate`| `IWebBrowser2`$\longrightarrow$`navigate2` $\longmapsto$`decline`	Block untrusted plug-ins' attempts to access current URLs and documents through calling the member functions `get_LocationURL`, `get_Document`, `navigate` and `navigate2` of `IWebBrowser2` interface.

Table 1. Examples of Security Policies.

(CPC). SpyShield allows trusted add-ons to communicate with a host application directly, which serves to limit performance degradation to untrusted add-ons. Selection of right CPC techniques can also reduce such overheads. For example, communication through shared memory is much faster than through pipes.

An important question is how to identify untrusted add-ons. SpyShield offers an automatic mechanism which classifies add-ons according to their hash values. The mechanism includes a database of hash values for trusted add-ons which are computed using a secure hash function such as SHA-256. An add-on is deemed untrusted if its hash cannot be found from the database. The content of the database can be maintained automatically using some heuristic rules: for example, the add-ons directly installed from a CD or signed by a trusted vendor such as Adobe Acrobat are considered to be trusted, while those downloaded from untrusted websites are untrusted. In order to prevent spyware from adding itself into the database, the database is also protected by a kernel driver called *proxy guardian* (Section 2.3). An authorized user is allowed to add in other trusted add-ons after being authenticated by her password and passing a CAPTCHA test [27] which tells humans and programs apart.

2.2 Security Policies

We developed a simple access control model for SpyShield, called *sensitive zone*. An application is said to enter a sensitive zone if it starts to process sensitive data. Within that zone, security policies are used to specify the resources to which an untrusted add-on's access is allowed or denied. If denied, the privacy information within the resources is preserved in the sensitive zone.

Sensitive data can be automatically identified with the metadata generated from users' inputs. For example, the URLs or IP addresses of sensitive websites such as banks are used to indicate the presence of confidential data like passwords and account balances. Other examples include names and directory paths of sensitive documents, email addresses and subjects of sensitive messages and keywords such as "password" within a data record. SpyShield can offer default settings of such metadata, which includes, for example, all banks' URLs. Authorized users are allowed to modify them.

Data imported by a host application are first checked by the reference controller against the metadata to determine whether a sensitive zone has been entered. An affirmative answer triggers the enforcement of a set of policies to restrict untrusted add-ons' access to such data. A security policy can be defined over add-on interfaces, their methods and input parameters to these methods. Table 1 gives example rules, which have controlled malicious IE plug-ins successfully in our expriements.

The security policies of a sensitive zone are applied to all the members in that zone. For example, if we include all banks' URLs in the same zone, the access control proxy will enforce the same set of rules whenever a browser visits any one of them. Flexibility and fine-grained controls can be achieved through multiple zones, which users are allowed to define. SpyShield offers a friendly and application-specific interface for authorized users to define sensitive zones and describe security policies. We present an example in Figure 2.

2.3 Proxy Guardian

Without proper protection, the access-control proxy is subject to a variety of attacks. For example, a spy add-on may tamper with the proxy's code and data, in particular sensitive zones and the hash database for trusted add-ons. Under some operating systems (OS) such as Windows, an add-on may also be able to read and write the virtual memory of its host application's process through API calls even when it is running inside another process [20]. To defeat these attacks, we developed *proxy guardian*, a kernel monitor to provide kernel-level protection to SpyShield components.

Proxy guardian interposes on the system calls related to file systems (e.g., `NtWriteFile`), auto-start extensibility points (ASEP) [29] such as registry keys (e.g., `NtSetValueKey`) and processes (e.g., `NtWriteVirtualMemory`), which enables it to block the attempts to access the proxy. Specifically, it ensures that only a dedicated uninstaller can remove the proxy's executables and data, and the ASEP for loading it to the memory. The uninstaller itself is also under the protection and can only be activated through both password authentication and a CAPTCHA test. Only the proxy is allowed to change its data. User-mode processes are prevented from accessing a host application's process image which also includes the reference controller. In addition, proxy guardian can keep other system resources related to SpyShield, such as DNS resolver, from being hijacked by spy add-ons, though the same protection can also be achieved by proper setting of untrusted executables' privileges through the OS.

Once an attacker manages to get into the kernel, it can directly attack proxy guardian. Such a threat can be mitigated by intercepting the system calls for loading a kernel driver to check the legitimacy of the code being loaded. A trusted driver can be identified by comparing its hash values with those of known reliable code, or verifying a trusted third party's signature it carries. This is a reasonable solution because kernel drivers are not as diverse as user-mode applications. Actually, many of them are standard and well-known, and their hash values are easy to obtain. This approach, however, cannot prevent spyware from getting into the kernel through exploiting a legitimate driver's vulnerabilities, for example, overrun of a buffer. Countermeasures to this attack must sit outside the OS, which we plan to study in the future research. Here, we just assume that kernel drivers are reliable.

Another functionality of proxy guardian is to make the existence of the access-control proxy transparent to the user and other applications. As an example, SpyShield can be installed on Windows as a normal plug-in, without changes of other plug-ins' registry keys; when an IE browser is trying to load untrusted plug-ins, proxy guardian blocks its system calls and lets the plug-in manager load them instead. This also allows an authorized user to easily turn off the proxy by leaving the loading procedure unchanged. We can further apply the techniques used by kernel-mode rootkits to manipulate the interactions between untrusted add-ons and the OS so as to hide the proxy's process, which protects it from being detected by spyware.

3 Implementation

To study the effectiveness of SpyShield, we implemented a prototype for Internet Explorer under Windows XP using C++. The choice of IE as the host application is due to the fact that the vast majority of known spy add-ons are in the form of IE plug-ins. However, our design is general, which also works for other applications such as Mozilla Firefox. In this section, we first present the technical backgrounds of COM and IE plug-ins, and then describe the details of our implementation.

3.1 IE Plug-in Architecture

COM Interfaces. The Component Object Model (COM) [31] is an extensible object software architecture for building applications and systems from the modular objects supplied by different software vendors. An object is a piece of compiled binary code that exposes some predefined services to COM *clients*, the service recipients. These services are offered through a set of COM interfaces, each of which is a strongly-typed contract between software objects to provide a collection of functions (aka., methods).

COM supports transparent cross-process interoperability which allows a client to communicate with an object regardless of where it is running. This is achieved through a system object encapsulating all the "legwork" associated with finding and launching objects, and managing the communication between them. When a client is accessing an object outside its process, COM creates a "`proxy`" which implements the object's interfaces. The "`proxy`" acts as the object's deputy by forwarding all the function calls from the client, marshalling all parameters if necessary and delivering the outcomes of the calls to the client. The remote process also accommodates a "`stub`" to mediate the communications between the "`proxy`" and the object.

Browser Helper Object and Toolbar. A browser helper object (BHO) is a COM object designed to expand the functionality of IE as a plug-in. A BHO object is required to implement the `IUnknown` interface, `IObjectWithSite` and `IDispatch` if it needs to subscribe to IE events during runtime. A toolbar is also a COM object serving as an IE plug-in. Compared with a BHO, it implements more interfaces to provide more functionalities which include graphics, usually in the form of a tool band, for a richer display and control for user interactions. A toolbar must carry four interfaces, `IUnknown`, `IObjectWithSite`, `IPersistStream` and `IDeskBand`, and may also involve several other interfaces such as `IInputObject` for focus changes of a user input object and `IDispatch` for event subscription and processing.

3.2 The Access-control Proxy

We implemented SpyShield as an access control proxy for IE plug-ins. The proxy includes a reference controller (RC) and an add-on manager (AM), two proxy components for managing BHOs and toolbars. The reference controller is a special plug-in which serves as both BHO and toolbar. It also contains an access control module to identify the sensitive zone being entered and thus to permit or block function calls originated from the browser and the add-on manager in accordance with security policies. The add-on manager acts on the behalf of the IE browser to provide COM interfaces to the untrusted plug-ins and mediate their communications. During the initialization stage, the browser loads trusted plug-ins and the reference controller only, leaving the task to import untrusted plug-ins to the add-on manager. This is achieved transparently through a kernel driver, which we describe in the next subsection. We implemented both one-process and two-process solutions, though here we only elaborate the second approach in which the add-on manager is running as a separate process.

Each proxy component contains three COM objects, *proxy BHO*, *proxy toolbar* and *proxy browser*. Proxy BHO/toolbar exports the COM interfaces on the plug-in's side to IE browsers and the reference controller. Proxy browser exports the COM interfaces on the browser's side to the add-on manager and untrusted plug-ins. These COM objects work in a collaborative way: for example, if one of them acquires the access to the `IUnknown` interface of an external object such as an IE browser, it passes the pointer of the interface on to the other objects, which enables them to directly interact with that external object. The reference controller uses its proxy BHO/toolbar as the delegate of untrusted plug-ins to interact with browsers, and the add-on manager employs its proxy browser as a substitute for the browsers to communicate with untrusted plug-ins. The other COM objects only serve to exchange parameters and requests with their counterparts in the other proxy object, and therefore are not used in our implementation of the one-process solution.

In the follow-up subsections, we elaborate our implementation of proxy interfaces, cross-process communication and access control mechanism.

Proxy Interfaces. Proxy browser implements a set of COM interfaces that an IE browser uses to accommodate BHOs and toolbars, and proxy BHO/toolbar adopts the interfaces on the plug-in's side. These interfaces 'wrap' their counterparts so as to put access control in place. For example, IE first triggers `Invoke()` within our proxy's `IDispatch` interface in response to the occurrence of an event, which allows it to decide whether to contact the same interface of untrusted plug-ins to activate their callback functions. Another example is an attempt from a plug-in to read the HTML files downloaded by IE, which must go through the proxy's `IWebBrowser2` interface and is therefore subject to its control. The COM objects within our proxy can also simulate the behaviors of the objects they substitute. As an example, our proxy follows IE's handling of the `QueryInterface()` call which does not return to the caller the interface reference of `IInputObjectSite`.

A technical challenge to enforcing access control comes from COM functions' capability to pass interface pointers. Without a proper design, an untrusted plug-in may acquire through our proxy a pointer to an IE browser's interface for directly interacting with that interface, which bypasses access control. Our solution is to detect such an attempt within the proxy's interface functions and returns to the plug-ins the pointers to the substitutes of the requested IE interfaces. This was implemented in the following functions: `QueryInterface()` in Interface `IUnknown`, `QueryService()` in `IServiceProvider`, `get_Document()` in `IWebBrowser2`, and `Invoke()` in `IDispatch`. `QueryInterface()` is the first function queried by plug-ins about other interfaces. `QueryService()` can be used to get the interface pointers of `IWebBrowser2`, `IOleWindow` and `ITravelLogStg`. Of particular interest is `get_Document()`, which returns a pointer to a COM object inside IE containing the documents being downloaded. Our prototype creates a substitute of that object and selectively copies to it the content of documents in conformation with access rules. `Invoke()` adds to the complication by taking an interface pointer of IE's `IDispatch` as part of the input parameters for a plug-in's callback function. Our proxy parses such parameters and modifies the pointer to a local substitute.

Table 2 describes the interfaces that we implemented for the access-control proxy.

COMPONENTS	INTERFACES
Proxy Browser	`IUnknown`, `IWebBrowser2`, `IServiceProvider`, `IOleCommandTarget`, `IInputObjectSite`, `IOleWindow`, `IConnectionPointContainer`, `IConnectionPoint`, `IWebBrowser2`, `IOleWindow`, `ITravelLogStg`, `IHTMLDocument2`, `IOleObject`, `IConnectionPointContainer`, `IOleContainer`, `IMarkupServices`, `ICustomDoc`, `IOleWindow`,
Proxy BHO	`IUnknown`, `IObjectWithSite`, `IDispatch`, `IWebBrowser2`,
Proxy toolbar	`IUnknown`, `IObjectWithSite`, `IDispatch`, `IDeskBand`, `IPersistStream`, `IOleCommandTarget`, `IInputObject`, `IWebBrowser2`

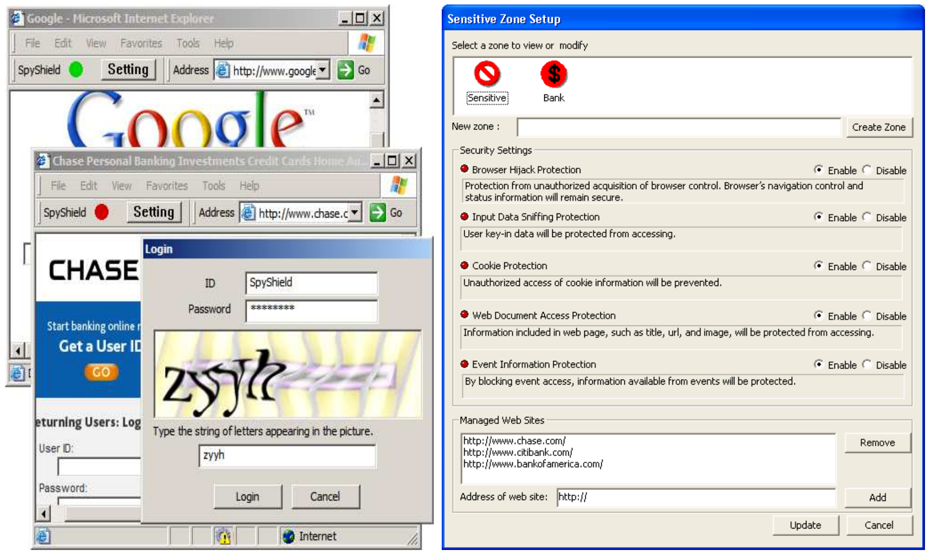
Table 2. Interfaces implemented in our prototype.

Cross-process Communications. As we introduced in Section 3.1, COM provides a mechanism which allows a client to request and receive services from an object running in another process through the interactions between the object's "`proxy`" in the client process and "`stub`" in its own process. This was employed by our implementation of the two-process solution to achieve cross-process communications (CPC). IE 6 offers the "`proxy`" and "`stub`" objects for all interfaces in Table 2 except `IInputObject` and `IInputObjectSite`. The problem has been fixed by IE 7 which provides `ieproxy.dll` to support CPC for both interfaces. Interestingly, we found this DLL can also be used in IE 6. Therefore, our prototype works under both IE versions.

Our two-process solution makes the add-on manager an independent process to accommodate untrusted plug-ins. This design, supported by COM's multi-threaded CPC, helps reduce the overheads of our approach in terms of memory usage: no matter how many IE processes have been launched, the add-on manager always stays in a single process. This is because COM automatically directs a new IE process's request to the existing add-on process which forks a new thread to serve it.

Access Control. The access control component was implemented in the reference controller. Whenever an IE browser visits a new website, the component acquires its URL from the parameters of `invoke` triggered by the event `DISPID_BEFORENAVIGATE2` and compares it with those defining sensitive zones. If the browser is found to be in one of the zones, corresponding security policies are applied. Otherwise, the proxy still needs to check the validity of the URL through a DNS query, as an invalid URL must also be protected to defeat error-page hijacking. Our prototype sets a default zone with the security rules in Table 1. To enforce the browser rule, the access-control proxy blocks IE's calls to untrusted plug-ins' `invoke` function. The plug-in rule was achieved by blocking the calls to `get_LocationURL`, `get_Document`, `Navigate` and `Navigate2` from the add-on manager. In addition, our kernel driver also intercepts and blocks the attempts to directly read or write the browser's virtual memory from another process.

Our prototype allows an authorized party to easily define a new sensitive zone and set security policies. It includes an IE toolbar to indicate the sensitivity of the current


Google - Microsoft Internet Explorer
File Edit View Favorites Tools Help
SpyShield Setting Address Go
Chase Personal Banking Investments Credit Cards Home Au...
CHASE
Start banking online
Get a User ID
GO
eturning Users: Log
Internet

SpyShield ● Setting Address http://www.google

SpyShield ● Setting Address http://www.chase.c

Login

ID: SpyShield

Password: ********

Type the string of letters appearing in the picture.

zyyh

Login Cancel

Sensitive Zone Setup

Select a zone to view or modify

Sensitive Bank

New zone : Create Zone

Security Settings

● Browser Hijack Protection [✓ Enable] Disable

Protection from unauthorized acquisition of browser control. Browser's navigation control and status information will remain secure.

● Input Data Sniffing Protection [✓ Enable] Disable

User key-in data will be protected from accessing.

● Cookie Protection [✓ Enable] Disable

Unauthorized access of cookie information will be prevented.

● Web Document Access Protection [✓ Enable] Disable

Information included in web page, such as title, url, and image, will be protected from accessing.

● Event Information Protection [✓ Enable] Disable

By blocking event access, information available from events will be protected.

Managed Web Sites

http://www.chase.com/
http://www.citibank.com/
http://www.bankofamerica.com/

Remove

Address of web site: http://

Add

Update Cancel

Fig. 2. The SpyShield toolbar.

website and provide an entrance to policy settings. Through that toolbar, an authorized user can access a friendly user interface (Figure 2) to view and modify existing sensitive zones and their policies, as well as add new ones. The simplest way to define a new zone is just to specify the URL of a sensitive website. The default security policies for a new zone decline all the requests from an untrusted plug-in whenever the browser is visiting that URL. To enable the user to set the policies with finer granularity, the interface offers the options to regulate a variety of channels through which a plug-in can access or even control the browser. For example, if 'Browser Hijack Protection' is enabled, the plug-in will not be allowed to invoke `Navigate2` which can be used to hijack the browser; if 'Cookie Protection' is set, the plug-in will be prevented from calling the COM functions such as `get_cookie` (in `IHTMLDocument2`) to acquire the cookie(s) associated with the website being visited.

To prevent spyware from tampering with the security policies, our prototype enforces a strict authentication which involves both password and a CAPTCHA. Figure 2 presents a screen snapshot of this mechanism. Such an authentication mechanism will only be invoked for customizing security policies, which is not supposed to happen frequently and therefore should not significantly increase users' burden. The chance for the setting change could be further reduced through careful construction of default zones, which can include the URLs of the sensitive websites, such as online banks.

3.3 Kernel Driver

We implemented proxy guardian as a kernel driver for Windows XP, which is used to prevent the add-on process from directly accessing the IE process, protect access-control data such as security policies and the database for trusted plug-ins from being sabotaged by spyware, and initialize the proxy transparently to avoid changes to IE and

CATEGORY	SYSTEM CALL
File system	NtWriteFile, NtDeleteFile, NtSetInformationFile
Registry keys and valuekeys	NtDeleteKey, NtRenameKey, NtReplaceKey, NtRestoreKey, SetInformationKey, NtSetValueKey, NtDeleteValueKey, NtQueryValueKey
Process, thread	NtTerminateProcess, NtTerminateThread
Virtual memory	NtAllocateVirtualMemory, NtReadVirtualMemory, NtWriteVirtualMemory

Table 3. System calls hooked in our kernel driver.

the Windows registry. This was achieved using an API hooking technique [26]. Table 3 lists the system calls hooked in our kernel driver.

The kernel driver can block the calls from the add-on process which operates on IE and the reference controller's virtual memory. System calls to modify the proxy's data are permitted only if they come from the proxy's process. The executables and the registry entry of the proxy can only be deleted and changed by an uninstaller, a program which is also protected by the kernel driver and allowed to run by authorized users only. Such users can revise the setting of the kernel driver, for example, specifying which files and processes should be under protection. We did not implement the mechanism to check the drivers to be loaded into the kernel, which can be done by interposing on other system calls.

The kernel driver can also insert our proxy between IE and untrusted plug-ins without altering any OS settings. It classifies the BHOs and toolbars recorded in registry entries[1] according to their hash values to compile a list of CLSIDs for trusted plug-ins. When an IE browser attempts to retrieve a plug-in's registry key, the driver intercepts its system call `NtQueryValueKey` and extracts the related CLSID. If it is not on the list, the driver removes it from the output of the call and notifies the add-on manager to load the plug-in instead.

4 Evaluations

We evaluated SpyShield using our prototype. Our purpose is to understand the effectiveness of our technique in containing spy add-ons and its overheads. All experiments were conducted on a desktop with Intel Pentium 3.2GHz CPU and 1GB memory. Its software includes Windows XP professional, Internet Explorer 7.0 and a vmware workstation. The effectiveness tests happened inside the virtual machine with a guest OS of Windows XP professional, and Internet Explorer 6.0. The performance of our prototype was evaluated in the host OS. We elaborate this study in the follow-up subsections.

4.1 Effectiveness

The effectiveness study aims at understanding SpyShield's ability to withstand spyware surveillance, which was achieved from the following perspectives. We first compared spy add-ons' networking behaviors in an unprotected browser with those under our prototype. Such behaviors usually constitute spyware's calling-home activities and contribute to the delivery of stolen data to the perpetrator. Therefore, this study reveals the effectiveness of our technique in preventing leakage of sensitive information. Then, we

[1] Specifically, the registry key for BHOs' CLSIDs is `\HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects`, two registry keys for toolbars' CLSIDs are `\HKLM\SOFTWARE\Microsoft\Internet Explorer\Toolbar` and `\HKCU\Software\Microsoft\Internet Explorer\Toolbar\WebBrowser`.

identified the COM events and calls being blocked by our access-control proxy. This further demonstrates the role SpyShield played in disrupting spyware surveillance, as these events and calls were used by spy add-ons to access sensitive data within an IE.

We evaluated our prototype using nine real BHOs and toolbars which are listed in Table 4. Five of them are spy plug-ins and the rest are legitimate. Under SpyShield, these plug-ins worked properly outside a sensitive zone. This demonstrates that our design does not disrupt plug-ins' legitimate operations. In the experiment, we first installed them to the unprotected IE inside the vmware station, and navigated the IE to access six websites listed in Table 5. In the host OS, we ran Wireshark [5] (aka. Ethereal), a traffic analysis tool, to record all network traffic from the virtual machine. Then, we activated SpyShield and repeated the above experiment.

INDEX	PLUG-IN	TOOLBAR	BHO	TYPE
1	AvenueMedia/Internet Optimizer	No	Yes	Spyware
2	Browser Accelerator	Yes	Yes	Adware
3	eXactSearch Toolbar	Yes	Yes	Adware
4	Mirar Toolbar	Yes	Yes	Adware
5	UCmore	Yes	No	Adware
6	Google Toolbar	Yes	Yes	Normal
7	LostGoggles	No	Yes	Normal
8	Security Software Search Bar 1.01	Yes	Yes	Normal
9	Yahoo! Toolbar	Yes	Yes	Normal

Table 4. BHOs and Toolbars used in our experiments.

Alias	URL	Sensitive Zone?
bbc	`http://www.bbc.co.uk`	NO
msn	`http://www.msn.com`	NO
google	`http://www.google.com`	NO†
chase	`http://www.chase.com`	YES
citi	`http://www.citi.com`	YES
invalid	`http://an.invalid.url`	YES‡

Table 5. Websites used in our experiments. †We visited "`http://www.google.com`" to retrieve "*money + saving + account*", keywords interesting to spyware, so as to elicit their networking behaviors. ‡We also included "`http://an.invalid.url`" in a sensitive zone because it leads to the DNS error page which is intensively used by spy plug-ins to hijack an IE browser.

To identify the network traffic caused by a plug-in, we recorded *baseline*, the network traffic observed while surfing these six websites without any plug-in. We also developed an analysis tool to capture the packets generated by a BHO or a toolbar. This tool classifies packets according to their destination IP addresses: any address outside baseline was deemed as coming from a plug-in. Effective suppression of such traffic within a sensitive zone offers the evidence to the efficacy of our technique.

A problem is that multiple visits of the website with dynamic contents might yield different network traffic. This could mislead our approach into including legitimate packets. We tackled this problem through cleaning the output of the tool against a manually compiled list of legitimate destination IP addresses. On the list were 25 addresses, most of which were from `msn` and `chase`.

We also recorded to a log file all the function calls intercepted by proxy interfaces, which told the story about plug-ins' activities. For example, `Browser Accelerator` calls `get_Document` to retrieve an HTML document as soon as a browser downloads it; this call was blocked by our proxy when the browser was inside a sensitive zone.

Index	Plug-in	bbc	msn	google	Sensitive Zone chase	citi	invalid
1	AvenueMedia/Internet Optimizer	-	-	-	-	-	Exist
2	Browser Accelerator	Exist	Exist	Exist	Exist	Exist	Exist
3	eXactSearch Toolbar	-	-	-	-	-	Exist
4	Mirar Toolbar	Exist	Exist	Exist	Exist	Exist	Exist
5	UCmore	Exist	Exist	Exist	Exist	Exist	Exist

Table 6. Network traffic from BHOs and toolbars in an unprotected IE browser.

Index	Plug-in	bbc	msn	google	Sensitive Zone chase	citi	invalid
1	AvenueMedia/Internet Optimizer	-	-	-	-	-	-
2	Browser Accelerator	Exist	Exist	Exist	Exist*	Exist*	Exist*
3	eXactSearch Toolbar	-	-	-	-	-	-
4	Mirar Toolbar	Exist	Exist	Exist	-	-	-
5	UCmore	Exist	Exist	Exist	-	-	-

Table 7. Network traffic from BHO/Toolbars under SpyShield. *Only part of the traffic in Table 6 was observed, which is irrelevant to the sensitive websites visited.

Traffic Differential Analysis. We present in Table 6 and Table 7 the results of our differential analysis of plug-ins' networking behaviors, which demonstrates the effectiveness of SpyShield in suppressing leakage of sensitive information. Both tables report plug-ins' network traffic when an IE browser visiting the URLs in Table 5, with Table 6 for an unprotect IE browser and Table 7 for the browser protected by SpyShield. Among these URLs, the first three were not in a sensitive zone and the rest were.

The tables show that most spy plug-ins produced network traffic while visiting some URLs. In the experiment, we observed that the occurrence of such traffic was contingent on the availability of the information flows from the browser to the plug-ins. Through examining the content of the traffic, we further discovered that in many cases such traffic carried the URLs of the websites being visited. Our prototype controlled the plug-ins' interactions with the browser, which contributed to curbing such traffic inside the sensitive zone. Outside the sensitive zone, the traffic recorded in both tables is identical, which suggests that our prototype did not disrupt the plug-ins' operations. We elaborate our analysis of individual plug-ins' behaviors below.

Legitimate Plug-ins. Legitimate BHOs and toolbars do not collect information without users' consent. Therefore, they should not produce network traffic without being explicitly invoked, unless there is an agreement between the company distributing these plug-ins and the customers. In our experiment, we did not observe any networking behaviors from all four legitimate plug-ins (Plug-ins with indices 6,7,8,9 in Table 4).

Spyware Plug-ins. Our implementation blocked all events and function calls related to untrusted plug-ins when the browser was visiting a sensitive website like '`http://www.chase.com`'. This could also affect spy plug-ins' communication which serves to deliver the information stolen to the perpetrator. In our experiment, we did observe the change of their networking behaviors, which are discussed as follows.

– `AvenueMedia/Internet Optimizer` is a BHO which can hijack a browser by redirecting it to an advertisement website whenever an invalid URL `http://an.invalid.url` is encountered. The same technique could also be used to stealthily place a malicious site between the user and a sensitive website for eavesdropping on their communication. The BHO employs a special technique to detect an invalid URL: it subscribes to an event

DISPID_BEFORENAVIGATE2 occurring when a website is to be accessed, and can therefore use a DNS query to determine the validity of the URL even before the browser does. Such a trick does not work on SpyShield, as our approach also hooked that event to identify sensitive zones. In our experiment, we found that the BHO's network traffic in response to an invalid link disappeared under our prototype.

– Browser Accelerator extracts the information from the web page loaded in a browser and sends it to data.browseraccelerator.com. Under SpyShield, the packets responsible for such behaviors could not be observed once the browser was inside the sensitive zone. However, we still detected some packets destined to client.browseraccelerator.com which our approach did not eliminate. We studied the contents of these packets and found them having little bearing on the sensitive website. Moreover, the same packets were also recorded when the browser was outside the sensitive zone. This leads us to believe they did not contain any sensitive information.
– eXactSearchBar also intends to hijack the invalid link. It redirects the browser to an advertisement site http://www.bestoftheweb.cc/errorpage/?src=4040&url=an.invalid.url once an error page was loaded. Packets related to such behaviors did not show up in the sensitive zone when the browser was protected by SpyShield. Previous research [28] also reported other networking activities of the spyware, which however were not observed in our experiments. This might be due to the change of the spyware's behaviors.
– Mirar collects data from the web page downloaded by a browser and displays advertisements related to its contents. It also encrypts its network traffic using SSL. In our experiment, we found its networking behaviors disappeared within the sensitive zone when SpyShield was running.
– UCmore is a toolbar which forwards the URLs of websites being visited and other information such as time and cache data of the local host to users.ucmore.com. This activity was stopped by SpyShield when sensitive websites were being surfed.

Control of Sensitive Events and Malicious Calls. Within a sensitive zone, SpyShield is designed to block event notifications and function calls in accordance with security policies. This was evaluated in our experiment through analyzing the log exported by our prototype which recorded the dangerous behaviors of spy plug-ins being prevented by the access-control proxy. Here we elaborate this study.

All spy add-ons in our experiment took IE's IWebBrowser2 interface as an entrance to other interfaces, and also made intensive use of it to retrieve a browser's sensitive data. In addition, most of them subscribed to certain events to trigger the calls for accomplishing their missions. As an example, we list in Table 8 the COM function calls of Browser Accelerator invoked by the event DISPID_DOCUMENTCOMPLETE which indicates the completion of downloading a web page to a browser. Another example is AvenueMedia/Internet Optimizer which took advantage of the event DISPID_BEFORENAVIGATE2 to identify an invalid link, and then called stop() and Navigate2to redirect a browser to another website.

SpyShield prevented these plug-ins' malicious activities within a sensitive zone through blocking all event notifications issued by IE. Without such notifications, function calls driven by these events disappeared. For example, our prototype intercepted and denied access to invoke() for 104 events subscribed by UCmore when visiting the website http://www.chase.com, which stopped 6 calls used to collect information from the site. Though most spy plug-ins were event-driven, we also found

Interface	Function Call	Description
IWebBrowser2	get_Document()	Retrieve the interface pointer of `IDispatch` in an IE object for the active HTML document.
IDispatch	QueryInterface()	Query the interface pointer of `IHTMLDocument2`.
IHTMLDocument2	get_parentWindow()	Retrieve the interface pointer of `IHTMLWindow2` in an IE object which accommodates the active HTML document.
IHTMLDocument2	QueryInterface()	Query the interface pointer of `IOleObject`.
IOleObject	GetClientSite()	Get the pointer of an interface which maintains the information regarding the display location of an embedded object in an active HTML document.
IHTMLDocument2	QueryInterface()	Query the interface pointer of `ICustomDoc`.
ICustomDoc	SetUIHandler()	Set the pointer of a customized interface.
IWebBrowser2	get_LocationURL()	Retrieve the URL of the web page that IE is currently displaying.

Table 8. Function Calls of `Browser Accelerator` triggered by Event `DISPID_DOCUMENTCOMPLETE`.

two exceptions which were capable of collecting data from a browser without being triggered by any event. Specifically, both `Mirar` and `UCmore` spawned threads once initialized and used them to periodically poll the function `get_LocationURL()` for the URL to be visited. This malicious behavior was blocked by our prototype with the plug-in rule in Table 1.

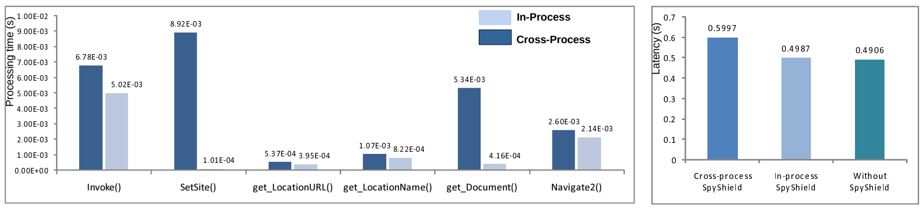
4.2 Overheads

We also studied the overheads introduced by SpyShield through experimentally evaluating the performance of plug-ins under our implementation (including the prototypes for both one-process and two-process solutions) against those running inside unprotected IE. Our research intends to understand the performance impacts of SpyShield from the following perspectives: (1) the overheads of cross-process communications, (2) the delay of COM function calls through the access-control proxy, (3) the waiting time of web navigation, a major feature of most IE plug-ins and (4) memory usage of the proxy. To this end, we conducted multiple experiments and also implemented a BHO which collaborated with our prototype proxy to record timing information.

Cross-process Communications. In this experiment, we measured the performance of cross-process communications and compared it with that of in-process communications. Our experiment involved emitting a message from our proxy through the COM interface to the BHO which bounced back a response. The round-trip delay during this process was halved and recorded by the proxy. This experiment was repeated for `1000` times each for the one-process setting in which the BHO and the proxy were inside the same process, and the two-process setting where the BHO ran in a separate process and the communication went through CPC. The results are the averages of the delays recorded in these experiments.

The average latency of CPC observed in the experiments is $177.3\mu s$, almost `1327` times as much as that of in-process communication which is merely $0.1336\mu s$. This result was echoed by a previous study [6]. Apparently, such a huge overhead could greatly affect the performance of the plug-ins running in a separate process, and therefore put the practicality of our approach in doubt. A close look at the time necessary for a plug-in to accomplish its missions, however, reveals that communication only plays a very small role. This suggests that the CPC overhead introduced by SpyShield could be overshadowed by plug-ins' other delays, which is confirmed by our studies on cross-process function calls and web navigation.

Cross-process Function Calls. We evaluated the performance of COM function calls both within a process and across the process boundary. Our experiments involved five COM functions extensively used by BHOs and toolbars, which include `Invoke` and `SetSite` on the plug-ins' side, and `get_LocationURL`, `get_LocationName`, `get_Document` and `Navigate2` on the browser's side. We used our proxy as a substitute for IE to invoke a BHO's function, so as to measure the time for completing that call. The delays of the calls on the reverse direction, from a plug-in to IE, were tracked by the BHO. Our experiments were conducted under both the one-process setting for in-process function calls and the two-process setting for cross-process calls. Figure 3.(A) describes the experimental results which were averaged over 10 experiments.



(A) The processing time of function calls in SpyShield

(B) The latency of webpage navigation

Fig. 3. The overheads of function calls and web navigation.

From the figure, we can see that the overheads of cross-process calls are not terrible: the processing time of most of them was between `21.5%` and `35.8%` longer than that of their in-process counterparts. The exceptions are `SetSite` and `get_Document`. `SetSite` sends the `IUnknown` pointer to a plug-in, which involves few other activities than communicating through the COM interface. Therefore, it is subject to the strong influence of CPC. Fortunately, the function is only invoked once during a plug-in's initialization and does not affect its runtime performance. Instead of CPC, the overhead for calling `get_Document` mainly comes from the delay for creating a substitute for an IE object in the proxy (Section 3.2). It is also one-time cost in most cases, as our proxy can re-use the substitute for subsequent calls to the function.

Web Navigation. The overhead of SpyShield is usually perceived by the user from the delay in receiving services from plug-ins. Most of such services require retrieving documents from the Internet. As an example, our study shows that web navigation is involved in at least `80%` features of `Google` Toolbar and 8 out of 9 features of `Yahoo!` Toolbar. Therefore, it is important to measure the latency of such a web activity in order to understand the performance of SpyShield.

In our research, we studied the delay caused by web navigation. Our experiment was carried out under the following three settings: (1) the BHO directly attached to an IE browser, (2) the BHO connected to the proxy within the browser's process and (3) the BHO and the add-on manager running in a separate process. In all these settings, the BHO directed the browser to the website `http://www.bbc.co.uk` by calling the function `Navigate2`, and recorded the time between the invocation of that function and the occurrence of the event `DISPID_DOCUMENTCOMPLETE` which indicates the completion of the navigation (i.e., all documents in the webpage have been downloaded). We repeated the experiment for 6 times under each setting to get the average latencies reported in Figure 3.(B): the navigation overhead was only `1.65%` for the one-process solution and `22.25%` for the two process solution. We believe such overheads are reasonable given the protection provided by our approach.

Memory Overheads. We also measured the memory overhead introduced by the two-process solution. The reference controller increased an IE browser's memory usage by `1MB`. The size of the memory allocated to the add-on manager varied with different plug-ins, which was around `18MB` for the `google` toolbar and `14MB` for the `Yahoo!` toolbar. On the other hand, we found that a `google` toolbar directly attached to IE added `4.8MB` to a browser's process memory. This became `3.3MB` for the `Yahoo!` toolbar. Therefore, the memory overhead of our prototype ranged from `11MB` to `15MB`.

Such an overhead is for a single browser window. As we discussed in Section 3.2, the add-on manager running in a separate process can provide services to multiple browser windows by spawning service threads. In our experiment, we observed that launching a new IE window only cost the add-on manager `0.1` to `0.5MB`, depending on the plug-in being requested. This is much lower than the memory cost of creating a new plug-in instance, which is necessary if the plug-in is directly attached to IE instead of the proxy.

5 Discussions

In this section, we discuss the limitations of the current design and implementation of SpyShield, and the potential improvement.

Limitations of Design. The current design of SpyShield is specific to the containment of spy add-ons. The user's interactions with sensitive data are still subject to the surveillance of keyloggers which intercept keystroke inputs, and screen grabbers which snoop on screen outputs. To defeat these attacks, we need to extend SpyShield to include system-wide security policies and an enforcement mechanism which prevents sensitive information from flowing into untrusted objects. Development of such a technique is part of our future research.

Although SpyShield can prevent spyware from being loaded into the kernel through system calls, it is unable to fend off the attacks through a kernel driver's vulnerabilities, for example, buffer overrun. When this happens, we rely on other techniques [16] to protect the kernel.

Limitations of Implementation. The current implementation of SpyShield applies the same security policies to the whole window object. This becomes problematic when a frame object is displaying multiple web pages in different zones within one window. A quick solution is to enforce the strictest policies of these zones. A better approach, however, should work on individual web page and treat them differently. Such functionality is expected in the future improvement of the prototype.

For simplicity, we only wrapped the COM interfaces requested by all the toolbars and BHOs used in our experiments. A thorough implementation needs to create all documented interfaces both in the reference controller and the add-on manager to accommodate different kinds of plug-ins.

6 Related Work

Existing defense against spyware infections mainly relies on detection techniques. These techniques are either based on signatures or behaviors, which we survey as follows.

Signature-based approaches analyze binary executables to identify spyware components or scan network traffics to detect spyware's communications with the perpetrator [2, 10, 23]. These approaches are fast, but can only detect known spyware. They can also be easily evaded [24]. Behavior-based approaches detect spyware according to its behaviors. Siren [13] and NetSpy [28] analyze the difference between the network traffic from an infected system and that of a clean system to identify spyware's networking activities. Web Tap [12] runs an network-based anomaly detector to capture spyware's network traffic. Gatekeeper [29] monitors the changes of Windows auto-start extensibility points for detecting spyware. GhostBuster [30] exposes rootkits by comparing a view of a clean system with that of an infected system. Recently, Kirda, et al proposed a technique [19] which applies dynamic analysis to detect suspicious communications between an IE browser and its plug-ins, and then analyzes the binaries of suspicious plug-ins to identify the library calls which may lead to leakage of sensitive information. SpyShield complements these techniques by adding an additional layer of defense which protects the user's privacy even after the detection mechanisms have been compromised.

Most of the existing proposals for spyware containment have been limited to protecting confidential inputs such as passwords from keyloggers. Bump in the Ether [21] offers a mechanism which bypasses common avenues of attack through a trusted tunnel implemented using a mobile device. SpyBlock [17] evades the surveillance of the keyloggers inside a virtual machine by directly injecting users' passwords into the network traffic intercepted by the host. These approaches are not very effective to spyware add-ons which are already part of their host application and can not only directly access its sensitive inputs but also snoop on its sensitive outputs such as the bank account displayed in a browser. In addition, they need either additional hardware (mobile device) or heavyweight software (a virtual machine). Microsoft's Next-Generation Secure Computing Base proposes encrypting keyboard, mouse input, and video output [8]. Though a promising approach, it significantly modifies current operating systems and its practicality is yet to see. By comparison, SpyShield is fully compatible with existing systems and can be easily installed and removed.

Similar to the two-process solution of SpyShield, *privilege separation* [25] partitions a program into a *monitor* to handle privileged operations, and a *slave* to perform unprivileged operations. Program partition is traditionally done manually over source code. Recent research, however, has made an impressive progress on automating this step [14]. While apparently assuming the same architecture, SpyShield actually aims at a different goal, inhibiting sensitive information from flowing into untrusted add-ons. To this end, it needs not only to segregate the privileged part of the program from the unprivileged part, but also to enforce security policies to their communication channel, the add-on interfaces, so as to regulate the information exchange between them. In addition, SpyShield separates a binary executable from its binary add-ons along their interfaces while privilege separation usually works on source code.

Another proposal which also employs the two-process architecture for privacy protection is *data sandboxing* [18]. The approach partitions a program into a private part which is allowed to access local files but forbidden to make network connections, and a public part which is permitted to perform networking activities but disallowed to read local data. Such a policy is enforced through system-call interposition [18]. In contrast, SpyShield aims at control of the communications through add-on interfaces, a task which system calls may not have sufficient granularity to handle.

Information flow analysis started with the famous Bell-LaPadula model which controls the interactions between processes and files [11]. More recent work [22, 32] focused on tracing data flows within a program. By comparison, SpyShield does not work on such instruction-level tracing, which incurs large performance overheads in absence of source code, and instead manages the information flows across the boundary between add-ons and their host application.

7 Conclusions and Future Work

In this paper, we propose SpyShield, a novel spyware *containment* technique, which can automatically block the visions of untrusted programs in the presence of sensitive information. Such a technique can also defeat the surveillance of new strains of spyware. As a first step, our approach offers general protection against spy add-ons which constitute a significant portion of existing spyware infections. SpyShield enforces security policies to add-on interfaces and other channels used by add-ons to interact with their host applications, so as to prevent sensitive information from flowing into untrusted add-ons. It can also defend itself against a variety of attacks. We implemented a prototype for protecting Internet Explorer and empirically evaluated its efficacy. Our experimental studies show that this technique can effectively mitigate the threats of spyware surveillance and also introduces a small overhead.

References

1. Browser extensions. `http://msdn.microsoft.com/workshop/browser/ext/extensions.asp`.
2. The home of spybot search & destroy. `http://www.safer-networking.org/`.
3. Mozillazine: Extension development. `http://kb.mozillazine.org/Dev_:_Extensions`.
4. State of Spyware Q2 2006, Consumer Report. `http://www.webroot.com/resources/stateofspyware/excerpt.html`.
5. Wireshark. `http://www.wireshark.org/`.
6. DCOM technical overview. `http://msdn2.microsoft.com/en-us/library/ms809340.aspx`, 1996.
7. XPCOM Part 1: An introduction to XPCOM. `http://www-128.ibm.com/developerworks/webservices/library/co-xpcom.html`, 1996.
8. Microsoft Next-Generation Secure Computing Base - Technical FAQ, July 2003. `http://www.microsoft.com/technet/archive/security/news/ngscb.mspx?mfr=true`.
9. Ucmore toolbar, the search accelerator. `http://www.ucmore.com/`, 2007.
10. Snort developed by sourcefire. http://www.snort.org/, as of January, 2006.
11. D. E. Bell and L. J. LaPadula. Secure computer systems: Unified exposition and multics interpretation. MTR-2997, available as NTIS AD-A023 588, MITRE Corporation, 1976.

12. K. Borders and A. Prakash. Web tap: detecting covert web traffic. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 110–120, 2004.
13. K. Borders, X. Zhao, and A. Prakash. Siren: Catching evasive malware (short paper). In *IEEE S&P*, pages 78–85, 2006.
14. D. Brumley and D. X. Song. Privtrans: Automatically partitioning programs for privilege separation. In *USENIX Security Symposium*, pages 57–72, 2004.
15. M. Egele, C. Kruegel, E. Kirda, H. Yin, and D. Song Dynamic Spyware Analysis. Usenix Annual Technical Conference. USA, June 2007.
16. T. Garfinkel and M. Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *NDSS*, 2003.
17. C. Jackson, D. Boneh, and J. C. Mitchell. Stronger password authentication using virtual machines. In submission, Stanford University, 2006.
18. T. Khatiwala, R. Swaminathan, and V. Venkatakrishnan. Data sandboxing: A technique for enforcing confidentiality policies. In *ACSAC*, December 2006.
19. E. Kirda, C. Kruegel, G. Banks, G. Vigna, and R. Kemmerer. Behavior-based spyware detection. In *Proceedings of 15th USENIX Security Symposium*, August 2006.
20. V. Mani. Cross Process Subclassing. `http://www.codeproject.com/dll/subhook.asp`, 2003.
21. J. M. McCune, A. Perrig, and M. K. Reiter. Bump in the ether: A framework for securing sensitive user input. In *Proceedings of the USENIX Annual Technical Conference*, pages 185–198, June 2006.
22. J. Newsome and D. X. Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In *NDSS*, 2005.
23. V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31(23-24):2435–2463, 1999.
24. S. Rubin, S. Jha, and B. P. Miller. Automatic generation and analysis of nids attacks. In *ACSAC*, pages 28–38, 2004.
25. J. H. Saltzer. Protection and the control of information sharing in miltics. *Communications of the ACM*, 17(7):388–402, July 1974.
26. S. B. Schreiber. *Undocumented Windows 2000 Secret: a programmers cookbook*. Addison-Wesley, May 2001.
27. L. von Ahn, M. Blum, N. J. Hopper, and J. Langford. CAPTCHA: Using Hard AI Problems for Security. In *EUROCRYPT*, pages 294–311, 2003.
28. H. Wang, S. Jha, and V. Ganapathy. NetSpy: Automatic Generation of Spyware Signatures for NIDS. In *Proceedings of ACSAC*, 2006.
29. Y.-M. Wang, R. Roussev, C. Verbowski, A. Johnson, M.-W. Wu, Y. Huang, and S.-Y. Kuo. Gatekeeper: Monitoring Auto-Start Extensibility Points (ASEPs) for Spyware Management. In *USENIX LISA 2004*, 2004.
30. Y.-M. Wang, B. Vo, R. Roussev, C. Verbowski, and A. Johnson. Strider ghostbuster: Why it's a bad idea for stealth software to hide files. Technical Report MSR-TR-2004-71, Microsoft Research, 2004.
31. S. Willliams and C. Kindel. The component object model: A technical overview. `http://msdn2.microsoft.com/en-us/library/ms809980.aspx`, Oct. 1994.
32. W. Xu, S. Bhatkar, and R. Sekar. Taint-enhanced policy enforcement: A practical approach to defeat a wide range of attacks. In *Proceedings of the 15th USENIX Security Symposium*, Vancouver, BC, Canada, August 2006.