Multi-Domain Sandboxing: An Overview

Robert Fischer
and
Ming-Yang Kao

TR-05-00

Computer Science Group
Harvard University
Cambridge, Massachusetts

Multi-Domain Sandboxing: An Overview

Robert Fischer* Ming-Yang Kao†

22nd September 2000

Abstract

In today's computing world, computer code is most often developed on one computer and run on another. Code is increasingly downloaded and run on a casual basis, as the line between code and data is blurred and executable code is found in web pages, spreadsheets, word processor documents, etc.

Not having the knowledge or resources to verify the lack of malicious intent of that code, the user must rely on here say and technological solutions to ensure that casually downloaded code does not damage his computer or steal his data.

Building on the past concepts of sandboxing and multi-level security, we propose multi-domain sandboxing. This security system allows programs more flexibility than traditional sandboxing, while preventing them from malicious actions. We propose applications of this new technology to the web, increasing the functionality and security possible in web applications.

1 Introduction

We focus on the security problems encountered when computer code developed by one party, but run on another party's computer. This type of sharing is very common today, happening when:

- A commercial program is installed off a CD-ROM.
- A shareware program is downloaded and installed off the Internet.
- A web page enhanced with Java, JavaScript, VBScript, ActiveX, etc. is viewed.
- A message is passed in a mobile agent system.
- A Microsoft Office document is loaded into a Microsoft Office application.
- An e-mail message enhanced with VBScript is viewed with Microsoft Outlook.

In some of the above cases, the user explicitly loads and runs a piece of software from a known software author; in others, the user may not even be aware that code is being downloaded and run locally on his machine. In either case, we refer to the computer code as *Mobile Code*, emphasizing the fact that it being run by someone other than the author, on a computer belonging to someone other than the author. The security problems, of course,

*Division of Engineering and Applied Sciences, Harvard University, citibob@eecs.harvard.edu
†Electrical Engineering and Computer Science, Tufts University, kao@eecs.tufts.edu

stem from the possibility that the code being run might destroy data or compromise security in some other way.

In the past, mobile code was shared infrequently and in a controlled fashion: typically, a small number of software packages, written by one of a few well-known vendors, were installed on any given computer system. Although such programs could theoretically destroy data on the system, commercial software vendors had incentives to ensure that their product did not. Even so, care was taken to limit the introduction of "Trojan Horse" and "Virus" programs via magnetic media.

The security problems associated with mobile code have increased in three ways as computers have become connected to the Internet:

1.1 Casual Downloads

It has become much easier and more common to casually download and run software. With the proliferation of interpreted languages, data formats with embedded code and e-mail attachments, the distinction between software and data has been blurred. No longer is code downloaded only when a user FTP's an executable and installs it; code also comes in e-mail, spreadsheets, word processor documents, web pages, etc. Users often do not even know when they are reading data and when they are executing downloaded code.

1.2 Viruses vs. Worms

In the past, a malicious piece of software, known as a virus, typically attached itself to a floppy disk or an executable program. It could not spread until the medium upon which the program was stored was physically moved to another computer, and the program run.

With the Internet, malicious software has more ways to spread itself than before. Known as a worm, a malicious piece of software can search for available computer systems compromise their security, and replicate immediately. The end result is that while a virus could take months or years to spread, a worm can spread to nearly all vulnerable computers worldwide in a matter of hours.

The "Internet Worm" of 1988 [9] first demonstrated the potentially costly and devastating effects now possible via the Internet. Most recently, the "Love Bug" of 2000 replicated at a similar speed. Worms that compromise even a small fraction of the world's computers can cause billions of dollars of damage.

1.3 New Malicious Possibilities

Last but not least, malicious code today can potentially cause much more damage than in the past. On a non-networked computer, the worst thing that malicious code can do is destroy all the data on the system. If daily backups are kept, this will at worst destroy a day's worth of updates.

Although the loss of a day's worth of work could cause great havoc in any corporate office, it is not as bad as what is now possible. An Internet-connected malicious program can choose not to destroy data, but to steal it. In this case, the program snoops around on the computer on which it's running, and sends whatever information it chooses back to the author, via the Internet. Such a program is both much harder to detect, and potentially much more damaging, than one that simply destroys data.

The theft of data and invasion of privacy is not just an idle concern; it has already happened. RealNetworks [20] and Microsoft [18], for example, have both been caught writing software that transmits private information back to the corporate network; for more information on such privacy abuses, see [1]. It is unclear which or how many other programs steal data in this fashion; many of them can be distributed and used widely before their malicious intent is discovered. The difficulty in discovery adds incentives for even reputable software vendors to write such code.

1.4 Focus: Data Theft

In this paper, we focus specifically on the problem of data theft from personal computers by mobile code. Because we are primarily concerned with theft across the network, we do not address classical security issues, which involve the security of data in the face of untrusted parties on the same computer. With a few exceptions, we assume data stored on a personal computer is considered private and should not be transmitted by the application program.[1]

We review past technical and non-technical solutions to data theft, focusing on the trade-offs involved between security on the one hand and ability for the application program to do its job on the other. We show why the solutions available today are not yet adequate to protect the user in today's networked environment.

We then present the concept of *Multi-Domain Sandboxing*, a security architecture that prevents untrusted mobile code from stealing data, while still allowing it access to the data on the local system, even private data! Building on that basis, we describe some exciting applications of multi-domain sandboxing. We expect them to be most useful in the development of client-server systems in which:

1. Significant amounts of data that must be kept private are stored on the client.
2. The client does not trust the code provided by the server's owners.

2 Security Solutions for Mobile Code

A number of security measures exist today; they can be evaluated along the following criteria:

1. **Security**: Do the security measure prevent data theft?
2. **Power**: What class of useful, legitimate programs can and cannot be run under this security measure?
3. **Workability**: Does the security rely on knowledge which the typical end-user, or even the typical power-user, may not have?

We proceed to examine past security systems and techniques, evaluating them against our criteria.

[1] For reasons of accessibility, the private data could just as easily be stored on a trusted remote server, rather than on the personal computer. Our analysis and arguments do not change in this case.

2.1 Paranoia

By paranoia, we refer to a reluctance by the end-user to engage in activities that might possibly involve the introduction of malicious code: typically reading e-mail, opening e-mail attachments, and installing software. Paranoia is widely used by corporate end-users, with the threat that the employee who introduces a virus or worm on his system will suffer a reprisal or even endure a complete system re-build from the sysadmin.

We consider here the common paranoia today:

2.1.1 Don't Install Software

Due to the possibility of malicious software and the instability of today's popular operating systems, many computer users are forbidden to install software on their computer. Instead, software, when it is installed, is done so by knowledgeable system administrators.

On the outset, this solution looks relatively secure, since mobile code is never introduced onto the system. However, mobile code *is* introduced whenever the system administrators install software, potentially compromising the security of the scheme. Although the system administrators may have better knowledge about what software might be malicious, they ultimately no more able to verify the security of megabytes of code than are end users; thus, the system is not workable. Finally, this scheme seems on the outset to be powerful; however, how much power is there to run useful software if users are afraid to install it?

2.1.2 Don't Read E-mail from Strangers

Microsoft's official policy in response to the "Love Bug" includes advising that users do not open "suspicious" e-mail attachments from unknown sources [21]. Unfortunately, this "advice" only adds to paranoia without enhancing security. The main reason it cannot guard against future worms like the Love Bug is that such worms are typically received as an e-mail attachment from a friend, not a stranger. It is not workable; users often do not know whether or not an e-mail attachment is suspicious.

Therefore, this scheme does not enhance security at all; however, it diminishes power because end-users become afraid to open even non-executable e-mails (often, the e-mail client does not make the difference clear to the end user). It is not workable, because very few people have memorized the list of the dozens of executable e-mail attachment formats around today. In conclusion, Microsoft's Love Bug advice seems to be little more than a desperate public relations ploy.

2.1.3 Conclusion

In conclusion, paranoid solutions add little to security, and detract greatly from power. They are also unworkable because they rely on specific knowledge that even experienced users may not have.

2.2 Trusted Sources

In a paranoid atmosphere, it is sometimes necessary to install software anyway. In these cases, the user might trust some sources of software more than others. For example, software from Oracle Corp. might be considered "safe" and therefore be installed, while shareware from an unknown author might be considered suspicious and is left alone.

Cryptographic technology, in the form of *signed code*, has been used to help users determine the author of the code they are running [11]. It works by attaching a digital signature which uniquely identifies the author of the code. Presumably, the user will install code verified as coming from well-known trusted sources, and reject code coming from unknown sources.

In analyzing this solution, one must not be led astray by the high-tech cryptography involved; simply using cryptography does *not* ensure security of any kind. In this case, the cryptography provides information to the user regarding the origin of the software. We analyze this solution by considering the cases where the software comes from a well-known vendor, and from a little-known author.

Suppose a user obtains a program from a well-known software vendor. This is generally done by downloading it from a well-known Internet site or by inserting a well-marked CD-ROM into the computer. In either case, even the novice user is relatively sure of the program's origins even without the use of digital signatures, because he trusts in the integrity of the well-known web site. Cryptography has not added information in this case, and the user's behavior remains unchanged: he installs software from a well-known vendor as before.

Now consider the user who has downloaded a program from a little-known shareware author. The program comes lists as author a name he's never heard of; the cautious user will avoid running that program because he knows nothing about its origins. Now suppose that the program comes with a digital signature. Using the digital signature, the author can verify that the program *really does* come from a source he's never heard of. The cautious user will refrain from installing such a program, with or without the signature!

Thus, digital signatures do not add useful new information in either case; it therefore does little over simple paranoia to add to security; it also does not affect power or workability for the same reason. Unfortunately, some users might be mislead to believe that the existence of a verified signature guarantees the safety of the code. In these cases, digital signatures ironically decrease security!

2.3 Sandboxing

Sandboxing refers to the practice in which potentially dangerous code is run in an environment that prevents it from carrying out dangerous actions. It is commonly used in cases, such as mobile agent systems [23, 15, 6, 16] and degenerate forms thereof such as Applets [7, 3], where code is transferred between computers on a casual basis and cannot be trusted. Prohibited activities typically include network communication, local I/O, and anything that could lead to network communication or local I/O.

When properly implemented and debugged, it is believed that sandboxes do prevent data theft. In their simplest form, they are also highly workable, because they require no configuration from the user. Unfortunately, sandboxes greatly diminish available power. The restrictions prevent applications from operating on the user's locally stored data. Since many useful programs in the personal computing world operate on user's data, sandboxing cannot be used to prevent data theft in a large number of cases.

2.4 Configurable Sandboxing

Seeing that a simple sandboxing system is too restrictive, some systems such as Java 2 [11] now offer configurable sandboxing. In this case, the user is given fine-grained control over

exactly which operations are and are not allowed for each application. Presumably, based on the user's varying level of trust in the code, he will allow different sets of permissions. Code which is trusted more will be allowed more privileged operations.

Unfortunately, such a system offers little security or power over a simple sandbox. Because all of the user's data can be stolen through just one security hole, there can be no continuum of trust in the world of security. The only applications which may be allowed even one dangerous operation can be those that are fully trusted. All other applications must be run in the most generous sandbox which still provides full security; presumably, the simple sandbox described above.

Moreover, configurable sandboxing is highly unworkable. Configuring a sandbox is a difficult, highly technical process that only security experts can get right. Simplifying the configuration is very difficult; even in simplified cases, the typical user does not know which settings he should choose. In this case, the user will likely configure the sandbox as directed by the software he is installing, and in this way fully trusting the software when he should not.

Configurable sandboxes do offer one potential advantage over simple sandboxes in terms of security and power. In some cases, a user may feel that he is willing to compromise some data on his system, but other data must be kept private. In this case, he can run untrusted software in a sandbox that only allows access to the his so-called *public* data.

2.5 Dynamic Sandboxing

Dynamic sandboxing is a variant of configurable sandboxing, in which the application program can request an increase in privileges while it is running. The user is free to grant or deny access. This technology, in combination with digital signatures, has been adopted by the Java Web Start system [2].

Upon closer analysis, this solution seems to be very much like a configurable sandbox in which the details of sandbox configuration are deferred to program run-time. It is unlikely that the user will gain significant knowledge between program installation and program run which would allow him to correctly configure the sandbox; if he were to gain such knowledge, he could usually change his statically configurable sandbox and re-start the program. Therefore, dynamic sandboxing might offer convenience over configurable sandboxing, but it does not increase security or power. It is also unworkable because it requires that the user make access control decisions in the middle of a program run, a time in which he is likely unable to gain the information needed to make an informed decision.

2.5.1 Server-Side Storage

Some application designers wish to build applications that work on a user's data, yet do not present the possibility of damaging the user's computer. The solution is to run the application program on a computer separate from the user's, and store the user's data there as well.

At first glance, this solution seems to offer high security: no code is ever run on the user's system, so it cannot corrupt or steal data on that system. Unfortunately, the solution requires that all data the user wishes the program to see is transferred to a remote server controlled by the authors. In this case, theft is not just possible, it is guaranteed! Such a system is only "secure" if issues of security apply only to locally stored data. Ironically,

the desire to protect the local computer has led to a large-scale invasion of privacy for the end-user.

Even so, applications using server-side storage have proliferated, particularly in the form of *web applications*. Other examples include downloadable office suites that store their data remotely. Although data stored remotely on someone else's computer is not at all private, factors of convenience and accessibility may lead some users to use such solutions in spite of the risks.

2.6 Capability-Based Systems

The idea of a sandbox is not new with Java; operating systems have restricted access to system resources since the 1960's. The technical details confining a process within a system so that it cannot communicate with outside processes were first examined in 1972 by Butler Lampson [17].

More recently, capability-based operating systems such as KeyKOS [6] and its successor Eros [22] have implemented these ideas systematically. In such a system, each process has a set of capabilities at its disposal; a capability is an unforgeable certificate required to access a system resource or service. Capabilities may be transferred to other processes via interprocess communication channels (which require the correct capabilities to use).

In this way, a capability system such as Eros is able to allow access control in a fine-grained fashion. Eros processes are allowed to transfer their capabilities to other processes, under certain circumstances, making it difficult to prove whether any given arrangement of processes and capabilities meets a system administrator's security criteria [13].

In summary, a capabilities-based system is similar to a systematic form of configurable sandboxing.

2.7 Conclusion

Although some of the above solutions provide enhanced security, in all cases, the typical end-user remains almost entirely vulnerable to malicious mobile code, especially code which seeks to steal data rather than destroy it. Sandboxing and its related forms has so far proven to be the most secure way to guard against data theft; however, a real tradeoff exists between data security and ability for the sandboxed application to use that data. Sandboxing has therefore been used only to secure small bits of "fun" code which do not operate on the user's data. Users who wish to run programs on their private data must still endure the risks of data theft.

3 Sandboxing Revisited

A closer analysis of the security issues reveals two types of sandboxes, both of which provide security to the user, which we name here:

- *Public Sandbox*: A program is prevented from accessing the local data store but allowed to communicate over the network. The worst such a program can do is broadcast the user's interactions with itself. If the user is warned that the running program has this capability, he can reasonably refrain from providing the application with data he feels is sensitive.

- *Private Sandbox*: A program is prevented from network communication but allowed access to the local data store. Such a program can do nothing worse than the Trojan Horse programs of the pre-Internet era. Security can be enhanced by allowing write permission only to a small portion of the data store. Ideally, a program should be allowed to write (and delete) only the data that it created. Even more restrictive, a program might be allowed to write, but never to delete.

In general, a program that is allowed only one of access to local data or access to the network can be trusted by the user to not steal data. A program can only steal data when it is allowed both local file access *and* network access.

Unfortunately for the simple sandboxing approach, many legitimate programs need both forms of access to function. These programs typically combine data from local and remote sources. For example, an e-mail client program needs to download messages from a POP server as well as display messages stored locally. A calendar client program needs to combine private calendar information of the local user with public calendar information from other users.

4 Multi-Domain Sandboxing

Although the above programs must be able to access local and remote data, in no case does the application need to send private data over the network; that would constitute theft of data. Therefore, we propose the technique of *Multi-Domain Sandboxing*. It allows a program access to the local data store and the network, but prevents the program from sending local data over the network. This effect is achieved by enforcing a partitioning of data within the program. It works as follows:

All data on which a program operates — stack variables, heap variables as well as long-term data — is separated into non-overlapping sets called *Security Domains*. In a simple case, data might be separated into two security domains, which we will call a "public" and a "private" domain. Data in the public domain is not secret and may be transmitted over the Internet; data in the private domain must not be made known to any external entities. Data must be prevented from moving from the private to the public security domain.

At any given time, a process is allowed to run in one of two sandboxes, each one associated with a security domain in this case. The public sandbox allows:

- read/write access to public security domain.
- access to the network.
- access to real-time information, such as a clock.

The private sandbox allows:

- read/write access to private security domain.
- read access to public security domain.
- NO access to the network.
- NO access to real-time information.

4.1 Switching Sandboxes

A program begins life running in the public sandbox. At any point, it can switch to the private sandbox by invoking the *CallPrivate()* primitive. This system call does the following:

- Switch system context to the private sandbox. Depending on the system implementation chosen, low-level context switching techniques, parameter marshaling, etc. may or may not be required.
- Run the specified function in the private sandbox.
- Return when the private-sandbox function completes. The function is not allowed to pass back any return value.

4.2 Covert Channels

Because the public sandbox has access to timing information and is informed when the function invoked by *CallPrivate()* completes, a *covert channel* can be set up in this case to pass data back from the private sandbox [17]. Although such covert channels have limited capacity, even small amounts of data theft can be damaging if the right data is stolen.[2] Therefore, we offer two refinements that serve to limit or stop the covert channel:

- The system can obscure the timing results by randomly running the the sandbox at varying speeds. This has the effect of increasing the noise on the covert channel, thereby decreasing its speed. It also has the effect of decreasing the speed of legitimately running programs. This technique is well-known in the literature [22].
- The system can obscure the accuracy of timing data by denying all access to the system clock and by randomly delaying network packets before they are delivered to the public function. As with the above obfuscation technique, this will result in decreased performance; however, network performance will degrade, rather than local system performance. This is much preferable: network performance is often already rather low, and its performance will not have to be degraded as drastically to limit timing accuracy.
- The *CallPrivate()* primitive can be modified to start the private function in a separate thread and return immediately; the function in the public sandbox is not notified when the thread completes. Of course, if the public and private portions of the application share a processor, a covert channel might still be set up if the public function can measure its own running time.

4.3 Special Capabilities

Depending on the application, the public and private sandboxes may provide specialized functionality to enable a program to work securely with a combination of public and private data. For example, the sandboxes designed for multi-domain Applets might provide primitives to write to the browser window under certain circumstances. More research must be applied to develop useful sets of sandboxes for important application areas.

[2] Recent press has focused on tremendous invasions of privacy possible over the Internet with just one cookie.

4.4 Conclusion

Thus, multi-domain sandboxing promises to provide an environment in which an untrusted program can both access local data and communicate on the network. It promises to be secure, in that it allows no theft of private data; it is powerful, because it allows a class of useful applications not previously allowed by simple sandboxing. Finally, it is workable because it requires no judgment on the part of the user. It is the only technology, proposed or implemented, that provides this combination of security, power and workability against data theft. Therefore, multi-domain sandboxing promises to be an important security technology in the face of mobile code.

5 Applications

We describe here the integration of multi-domain sandboxing into interesting and useful application platforms. We then describe useful application programs that could be built on these platforms.

5.1 Simple Application

Most simply, multi-domain sandboxing can be used when running typical software installed from an Internet download or a CD-ROM. Because of the technical aspects described above, software must be specially written to run in a multi-domain sandbox. Operating systems could support the technology to greater or lesser extent; alternately, sub-systems similar to the Java Virtual Machine could be developed.

Not all legitimate programs can be run an a multi-domain sandbox. For example, FTP by its very nature requires the ability to send local data over the network.[3] It is not yet clear what other programs can and cannot be run in a multi-domain sandbox.

Therefore, the user must be given the choice of running software with or without sandbox. Presumably, she will be more willing to run the software if it has been built to work in the sandbox.

5.2 Mobile Agent Computing

Mobile Agent computing refers to distributed computing in which the messages passed back and forth consist of pieces of code. When a message is received, it is run to obtain its desired effect. Any message-based system RPC-type system, such as HTTP, can be replaced by an equivalent agent-based in a straightforward manner [12, 8]. Although the most mobile agent systems are semantically equivalent to similar RPC-type systems, mobile agents have been found to offer two distinct advantages to RPC:

- They enable higher performance over low latency links: by uploading an intelligent controller program to the device being controlled remotely, network requirements are reduced.
- They enable higher flexibility.

For these reasons, some system designers choose to base their systems on the mobile agent paradigm. Because mobile agent systems involve the execution of untrusted code on

[3] FTP could, however, run well in a simple dynamic sandbox

all computers involved, data theft becomes a very real issue. All systems surveyed solve this problem with simple sandboxes, implemented by a variety of approaches [16, 15].

For many proposed mobile agent applications, the simple sandbox is not unduly constraining. In general, such applications involve large amounts of public data on a server; the mobile agent often sifts through this data, finding relevant data for the person who sent the agent. For example, a library catalog might allow mobile agents to search its database in arbitrary ways, sending back to the user only those entries that the agent deems important to the user. Depending on the number of entries through which the agent must sift, this could use vastly less network bandwidth than a semantically equivalent RPC implementation, in which data is sent to the user before it is analyzed and (mostly) discarded. Since the data being searched is public and is supposed to leave the server, a simple sandbox is sufficient to ensure the integrity of the data server.

Interestingly, mobile agents have not been proposed for applications that operate on private data; presumably, this is because of the security problems involved. Since personal computers are for the most part private, most mobile agent computing has involved running interesting mobile agents on server computers, not on clients.

However, one could imagine a mobile agent system in which mobile agents server purposes that require modification of local data on the client. For example, Microsoft envisioned the Windows Scripting Engine as a convenient way for system administrators to update entire networks of computers [5]. Typically, the system administrator will prepare a program that must be run on every computer. He then distributes it to all employees of the corporation via e-mail, asking each employee to run it by clicking on the attachment. Once invoked, the mobile agent performs the necessary system updates. Unfortunately, Microsoft did not use sandboxing, digital signatures, or any other type of security to ensure the safety of such mobile agents, leaving its systems open to a host of e-mail worms.

By replacing the simple sandboxes of mobile agent systems with multi-domain sandboxes, one could increase the power inherent in them, thereby expanding their applicability. Multi-domain sandboxes would allow mobile agents to intelligently read and modify certain local client-side data, without fear of data theft.

Interestingly, in contrast to the conclusions of Harrison et al [12], mobile agents in a multi-domain sandbox *do* offer something semantically different from RPC. This is because in an RPC system, the execution of remotely-available system calls involves the disclosure of data to the remote system. That is also true for the execution of exposed API calls in a mobile agent system if the agent is allowed network access. However, in the multi-sandbox case, it is possible to give the agent significant access to local data. The agent can operate on, add to, and even modify private local data. This is not possible in a secure fashion using RPC.

5.3 CliletWeb: Web Applications with Client-Side Storage

As mentioned above, the web application of today compromises a user's data security by storing all the user's data on a remote server. For some web applications, this is not a big problem; for example, an e-commerce web application needs to store each user's address on the server so it can know where to send the requested merchandise.

However, for some other web applications, the storage of data on the server is an unacceptable security risk. For example, a personal calendar application requires storage of private details regarding someone's life. Even the potential benefits of a networked calendar application — the sharing of public portions of one's calendar, automatic updates from

general public databases, etc — do not offset the potential risks for many people.

Therefore, although web applications offer many potential benefits over classic locally-installed applications — universal accessibility, easy collaboration with others, maintenance-free software updates — they are under-used because of the security risks involved.

Therefore, we propose the development of web applications that store certain data in a private data store on the client. The web application is able to read and manipulate this client-side data and even combine it with publicly available data off the network; but the server from which the web application originates cannot read it. Programs such as web calendars can store their data on the client, thereby ensuring the user's privacy.[4]

5.3.1 Implementation

Web applications with client-side storage can be implemented using a mobile agent system with multi-domain sandboxing. In this case, a web server can be modified to return executable code, instead of HTML, in response to an HTTP request. In order to produce the final HTML page displayed to the user, that code is run in a multi-domain sandbox inside the browser. The sandboxes provide a *writeBrowser()* function, enabling the received mobile agents to construct an HTML page on the client.

It is assumed that some of the data the user gives the web application will be public, and some private. Therefore, HTML is extended to allow for *public* and *private* form elements, which look visually distinguishable on the screen. When a user submits a form, only the public form elements are sent to the server. The private form elements remain on the client, awaiting processing by the mobile agent expected shortly from the server.

Since HTML written to the browser window is not transmitted to the remote server, it is safe for the private sandbox to allow writes to the browser. However, certain portions of an HTML page *do* have the potential to send code back to the server without the user's knowledge; these are anchors and hidden form elements. For this reason, the private sandbox is not allowed to write anchors or hidden form elements to the browser. This rule is enforced by filtering and parsing the output of the private portions of the mobile agent, in order to remove any violations.

In order to allow for client-side storage, the private sandbox is provided a facility with which to read and write private client-side data. Typically, each web application might be provided its own sub-tree in the file system; however, most web applications work more conveniently with relational databases than with filesystems. Therefore, each web application might also be provided with a relational database on the client computer, which it can set up and administer as it wishes. Size limits might be imposed on client-side storage by a web application, in order to prevent denial of service attacks.

We have implemented a proof-of-concept system allowing web applications with client-side storage. In order to show that it can work, we developed an address book application that stores its data entirely client-side.

5.3.2 Applications

We suggest here some applications of client-side storage in web applications, all of which can be implemented using the above architecture. The most intriguing examples involve

[4] In the case that the client computer is not universally accessible, this could reduce the universal access allowed by web applications. In this case, private data could be stored on a highly available computer trusted by the user.

the combination of private and public data on one screen.

On-Line Banking You are a customer at a bank. Every month, you look at your statements on-line via a web application. For each check you wrote in that month, the bank provides you, via a web application, with a line-item consisting of the check number and the amount of the check. You would like to annotate those line items, including a textual description of each check, a budgeting category, etc.

However, you value your privacy, and would like to store the annotations on your home computer. You have considered switching to a locally installed home finance application. Instead of looking at your account information on-line, you would download it periodically, and then annotate it off-line. Even if you trusted locally installed applications with your data, you realize that your home finance application does not allow the type of data integration you desire between local and remote data. For example, as information regarding a previously downloaded transaction is updated at the bank, you would like that update to display on your computer. Downloading all the bank's data at the beginning prevents this type of automatic update.

This banking application could be implemented easily as a web application with client-side storage. Information provided by the bank would be stored on the bank's server and downloaded on demand. Information provided by the user would be stored on the client computer. The mobile agents sent by the bank would be responsible for combining local and remote data in intelligent ways, querying remote data, and updating local data.

Web Calendars Web-based calendar applications present a similar problem. Schedule information generally consists of public data (such as office hours) and private data (such as vacation destinations). Users can gain great benefit by sharing their public calendar data; it is appropriate to store this on an essentially untrusted remote server. However, users wish to keep their private schedule information private.

It is important that each user's private information is combined with his public information, as well as other users' public information, so that he does not promise to be in two places at the same time. If the computer does not combine this information automatically, the user must do so manually.

As described in the banking example, a web-based calendar application that guarantees the privacy of private calendar data can be implemented as a web application with client-side storage. In this case, networking is used as a powerful tool for collaboration between users.

Inventory Control You run an auto repair shop; as such, you keep on hand an inventory of spark plugs. You run on your computer system an inventory-control system. You typically make new spark plug orders over the web, choosing your suppliers dynamically on the basis of price, convenience, etc. When you run low on spark plugs, the following currently happens: your inventory specialist browses the web to find an appropriate supplier. He places an on-line order, printing out the HTML receipt which returned via HTTP. He then enters the information from that printed receipt into the inventory control system.

You would like to eliminate the inefficient and error-prone step of typing data into the inventory control system. Ideally, when a purchase receipt is sent to your computer, you would like the computer to enter the information into the inventory system.

This system can be implemented by allowing the mobile agents originating from the e-commerce web application to write to the local inventory database. It raises some interesting problems of sharing: in previous cases, all data a web application dealt with was owned exclusively by that web application. In this case, all e-commerce applications must be able to write to the inventory database, which is owned by none of them. Presumably, the user has some locally installed program giving full inventory access.

This example also raises some standards issues. In order for an auto-parts inventory system to be useful, all local databases must implement the same API's for the mobile agents, and all e-commerce web applications must send mobile agents that conform to these API's. Each industry would need to develop its own mobile agent API specifications. Although this is a potentially large undertaking, it is not impossible. A variety of industry-specific XML templates, envisioned to allow easy data exchange of this sort, have already been developed and standardized; see [4] for more information.

Mail Client You subscribe to an ISP, which provides you with a remote mailbox of finite size. However, you are the type of person who likes to keep all your past e-mail. Therefore, from time to time, you download your e-mail to your home computer.

This downloading process presents a problem. When you are at home, you have access to all your e-mail, which has been downloaded. However, when you are away from home, you have access only to e-mail which you have not yet deleted off the ISP's server; typically only the past couple of weeks. Moreover, the e-mail client program you use on the road, generally a web-based system, works completely differently from the e-mail client you use at home, generally a locally loaded client program.

By implementing your e-mail client as a web application with client-side storage, you can have the best of all worlds. The application will search for e-mail on the server as well as e-mail on the client, displaying it as one integrated whole.

Interestingly, the act of viewing a message may be made equivalent to the act of downloading that message. When the mail server sends a message to be viewed, that message must be sent as a data structure inside a mobile agent. In addition to displaying the message on the screen, the mobile agent can store it on the client's computer. Therefore, no separate downloading step is needed in such a system.

Contrast this to classic web applications, in which HTML is essentially unreadable by automated means. Even if a piece of data has been transmitted to the client and displayed in its browser, very little else can be done with the data in that form. XML has been proposed as another way to increase the machine-readability of data involved in web applications.

5.4 Multi-Domain Graphical User Interfaces

A web application with client-side storage is interesting because it combines separate security domains side-by-side on a user's screen. In this case, the task is accomplished by the multi-domain sandbox, combined with some HTML magic.

HTML offers a convenient and easy way to build certain graphical user interfaces. More traditionally, GUI's are built using object-oriented systems such as SMALLTALK [10] or Java Swing [14]. It should certainly be possible to build an object-oriented GUI toolkit in which portions of any given application reside in the public security domain and portions in the private security domain. As in the case of web applications, user interface elements stored in the public and private domains would display differently, thereby properly alerting the user.

In a normal GUI system, interface components interact to handle and coordinate events. A GUI system that can run in a multi-domain sandbox would have to restrict inter-component interaction because of the sandboxes. It is not clear exactly how inter-component interaction would change in this case, or how the architecture of the GUI system would have to change. As with the web application, it is expected that functionality special to the GUI case would have to be built into the sandboxes, allowing the special types of communication required by GUI components.

5.5 Military Security

Military organizations generally classify their data into a set of non-overlapping security domains. Strict rules and procedures are set in place to prevent the flow of information from security domains of higher classification to lower classifications; entire computer systems implementing these rules have been built [19].

Multi-domain sandboxing was in part inspired by the military approach to security. There is great potential for military applications of the technology.

6 Conclusion

In today's computing world, computer code is most often developed on one computer and run on another. Code is increasingly downloaded and run on a casual basis, as the line between code and data is blurred and executable code is found in web pages, spreadsheets, word processor documents, etc.

We focus on the specific problem of data theft by mobile code. Not having the knowledge or resources to verify the lack of malicious intent of that code, the user must rely on here-say and technological solutions to ensure that casually downloaded code does not steal his data. Past solutions limit the utility of the computer for the average user, are not sufficient to guard against malicious intent, unnecessarily limit the capabilities of mobile code, and too often expect the user to provide information he does not have and cannot reasonably obtain. The end result is mass confusion and lack of protection from malicious intent, costing the corporate world billions of dollars each year in security breaches.

Building on the past concepts of sandboxing and multi-level security, we propose multi-domain sandboxing as a way to give programs greater power than possible with simple sandboxing, while still protecting private data on the host system. In particular, multi-domain sandboxing allows mobile code to securely access local data as well as communicate on the network, but not to communicate private local data over the network. We discuss how multi-domain sandboxes might be implemented and configured to guarantee security and minimize covert channels.

We then discuss three applications of multi-domain sandboxing. It can be used routinely when running most programs, it can be integrated into mobile agent systems to expand their capabilities. Most interestingly, it can be used to allow for secure client-side storage in web applications, in which client-side data is kept secret from the server.

In order to show that these concepts can work, we have demonstrated a simple web application that stores all its data on the client. We also mention a number of other interesting web applications that can be built with our technology. We expect it to be especially good in cases that require a combination of local and remote data, cases that require machine-readable web a pages, and cases that require local database updates in response to a web

page.

Finally, we propose that multi-domain sandboxing might be used to build more complex graphical user interfaces that securely combine private and public data on-screen in the same program. More research is required on the technical details.

6.1 Future Work

In the future, we plan to refine and implement the ideas presented here:

- We will develop a prototype multi-domain sandbox. It is not yet clear whether Java or some similar system, built with simple sandboxing in mind, can be used for the task. It is not clear how much domain-switching will cost in terms of performance, or how often it will be necessary.
- Building on past experience with web applications with client-side storage, we will build such a system, based on mobile agents and a multi-domain sandbox. We will focus on the sandbox features special to the web application problem. We will continue to develop innovative applications using client-side storage.
- We will analyze the inter-object communication inherent in GUI toolkits, allowing us to develop a design for a GUI toolkit that supports multi-domain sandboxing.
- We will quantify the bandwidth of the covert channels, and to develop the best way to minimize these channels.
- We will characterize the class of applications that can and cannot be run in multi-domain sandboxes. We will also characterize applications that run just fine in simple sandboxes.
- We will seek possible military applications of multi-domain sandboxing.

References

[1] . http://www.junkbusters.com.

[2] . Developer's Guide, Java Web Start, Version 0.4 Early Access Release. http://java.sun.com/products/javawebstart/docs/developersguide.html.

[3] . Frequently Asked Questions — Java Security. http://java.sun.com/sfaq.

[4] . The XML Catalog. http://www.xml.org/xmlorg_registry/index.html.

[5] . Post Love Bug, Microsoft Trades Flexibility for Security. *Business Week Online*, May 15 2000. http://www.businessweek.com/bwdaily/dnflash/may2000/nf00515d.htm.

[6] Alan C. Bomberger, Norman Hardy, A. Peri Frantz, Charles R. Landau, William S. Frantz, Jonathan S. Shapiro, and Ann C. Hardy. The KeyKOS NanoKernel Architecture. *Proceedings of the USENIX Workshop on Micro-Kernels and Other Kernel Architectures*, pages 95–112, April 1992.

[7] Mary Campione and Kathy Walrath. *The Java Tutorial Second Edition: Object-Oriented Programming for the Internet*. Addison-Wesley Pub Co, 1998. http://java.sun.com/docs/books/tutorial/index.html.

[8] David Chess, Colin Harrison, and Aaron Kershenbaum. Mobile Agents: Are They a Good Idea? – Update. In *Mobile Object Systems: Towards the Programmable Internet*, pages 46–48. Springer-Verlag, April 1997. Lecture Notes in Computer Science No. 1222.

[9] Peter J. Dennin, editor. *Computers Under Attack: Intruders, Worms and Viruses*. ACM Press, 1990.

[10] A. Goldberg. *Smalltalk 80: Creating a User Interface and Graphical Applications*. Addison-Wesley, 1983.

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

[12] Colin G. Harrison, David M. Chess, and Aaron Kershenbaum. Mobile Agents: Are they a good idea? http://www.research.ibm.com/massdist/mobag.ps, March 1995.

[13] Michael Harrison, W. L. Ruzzo, and J. D. Ullman. On Protection in Operating Systems. *Communications of the ACM*, 19:461–471, 1976.

[14] Cay S. Horstmann and Gary Cornell. *Core Java 2, Volume II*. Sun Micorsystems Press, 2000.

[15] Gunter Karjoth, Danny B. Lange, and Mitsuru Oshima. A Security Model for Aglets. *IEEE Internet Computing*, pages 68–77, July/August 1997. http://computer.org/internet.

[16] Kato, Toumura, Matsubara, Aikawa, Yoshida, Kono, Taura, and Sekuguchi. Protected and Secure Mobile Object Computing in PLANET. In *Proceedings of the 2nd ECOOP Workshop on Mobile Object SYstems*, pages 20–27, Linz, Austria, July 8–9 1996. `http://cui.unige.ch/~ecoopws/ws96/2.ps.gz`.

[17] Butler W. Lampson. A Note on the Confinement Problem. *Communications of the ACM*, 16(10), 1973.

[18] John Markoff. Microsoft to Alter Software in Response to Privacy Concerns. *New York Times*, March 7 1999. `http://www.nytimes.com/library/tech/99/03/biztech/articles/07soft.html`.

[19] Department of Defense. Department of Defense Trusted Computer System Evaluation Criteria, December 1985. DoD 5200.28-STD `http://www.radium.ncsc.mil/tpep/library/rainbow/5200.28-STD.html`.

[20] Sara Robinson. CD Software Is Said to Monitor Users' Listening Habits. *New York Times*, November 1 1999. `http://www.nytimes.com/library/tech/99/11/biztech/articles/01real.html`.

[21] Joel Scambray. Antisepsis for I LOVE YOU. *Microsoft TechNet*, June 2000. `http://www.microsoft.com/TechNet/security/au052200.asp`.

[22] Jonathan S. Shapiro, Jonathan M. Smith, and David J. Farber. EROS: a fast capability system. *Operating Systems Review*, 34(5):170–185, December 1999.

[23] Gatot Susilo. Infrastructure for Advanced Network Management based on Mobile Code. Technical Report SCE-97-10, Dept. of Systems and Computer Engineering, Carleton University, 1997. `ftp://ftp.sce.carleton.ca/pub/netmanage/sce-97-10.ps.gz`.