Security In Computational Electronic Mail Systems

Y.N. Sng , C. Quek and S. Foo

School of Applied Science, Nanyang Technological University,
Nanyang Avenue, Blk N4 #2A-36, Singapore 639798
Singapore 639798

Abstract

Computational electronic mail (ce-mail) is an electronic mail that contains an embedded program or interpretable script. Unlike conventional mails, ce-mails are active mails that execute on the recipients' machines. Ce-mails find applications in a wide range of areas including electronic forms, interactive informational mails and intelligent autonomous agents. Potentially powerful, ce-mails nonetheless carry high security risks as a result of malicious attacks that can arise from executing them. Security attacks can take the form of integrity, privacy and denial-of-service attacks on the systems. Such attacks may cause irreparable damages or unquantifiable costs to recipients.

This work proposes a number of techniques to provide protection without compromising too much of the power of ce-mails. In particular, the concept of safe virtual environments for the execution of ce-mails and the utilisation of safe ce-mail languages are highlighted. Safe virtual environments may be attained through the execution of ce-mails at a lower priority, or the creation of a separate disk partition or virtual machine for the execution of ce-mails. Two ce-mail languages, Safe-Tcl and Java, are evaluated and contrasted against the four main features of a safe ce-mail language that include security, power, extensibility and standardisation.

1. Introduction

Electronic mail (e-mail) systems have been widely used in industry, business, academia and homes as a useful tool for communication. However, text e-mails or multimedia e-mails are passive mail systems in that the information they display is static. Computational electronic mail (ce-mail) evolved as an enhancement to the current mail systems [6]. Ce-mail, which is alternatively referred to as Active Mail or Enabled Mail [5] contains an embedded program or interpretable script that is sent and executed or interpreted at the recipient's machine.

In the first instance, the ce-mail is compiled into an executable form at the sender's end, then transported to the recipient's end where it will execute upon activation by the recipient. In the second instance, it is expressed in an interpreted language (e.g. Tcl/Tk [7,8], Safe-Tcl [3], and JAVA [10]) and contained in a script file which requires no compilation at run time. The script file
is transmitted, loaded and executed at the recipient's end. The first approach requires the active mail be compiled for execution on the recipient’s machine that may have a different configuration and operating system. For this reason, the second approach is preferred. As long as appropriate interpreters are available on different users’ computers, executing a ce-mail poses no problem.

Typical applications of ce-mails include electronic forms, interactive information tools and autonomous agents [1][2]. Contrary to existing mail systems, ce-mail is potentially powerful as well as potentially hazardous. Data in the recipient's machine can be unlawfully accessed, destroyed or tampered, and machines can be hacked. Therefore, security in ce-mail systems becomes a prime issue in the delivery of such systems due to the risks of irreparable damages or unquantifiable costs to recipients.

2. Threats from executing ce-mail

It is a daunting task to list every possible threat from executing a ce-mail. Just like its applications, its threats are limited only by the hacker’s imagination. Once the ce-mail reaches the recipient’s machine, it becomes a trusted component in that machine. Therefore, what it does subsequently becomes more difficult to track because the underlying operating system cannot possibly monitor every trusted process running within its environment.

Such malicious ce-mails, or Trojan Horses, make use of this loophole to create havoc in the recipient machines. It enters the computer system posing as something harmless and trustworthy. Once it gains entry into the system, it performs acts which systems normally would not allow untrusted programs to do. These may include unlawful access to sensitive information, modifying data, or replicating itself to infect other machines as in the case of the Internet Worm [12].

Threats from ce-mail take many forms that can include the:

- Manipulation of the local file system on the recipient's machine. With access to the file system, data files can be retrieved, altered or deleted without the user’s knowledge or approval. Unwarranted write access to the local file storage system can also occur. Disastrous consequences arise as this can be used for sabotage, business espionage, intrusion of privacy or modification of password files.

- Manipulation of system properties that may include switching off the protection for file systems or password checks. Thereafter, other harmful actions on the file system become possible.

- Opening of an arbitrary network connection that will allow the malicious program to communicate directly with other networked machines thereby rendering further harm. The possibility of having direct read or write access to a file system across the network provides the malicious program greater power and flexibility as it does not have to reside physically on every machine to cause harm.

- Invocation of other programs residing on the recipient's machine. This poses potential harm that is many times more than what a ce-mail alone can achieve. Local programs that may be low-level and powerful, and trusted by the operating system may be activated to perform tasks that the ce-mail cannot do. The damage aggravates if the ce-mails are small and the only thing they do is to invoke other programs to create harm. Such ce-mails will be far more difficult to detect in the system or network.
• Arbitrary loading of libraries into the ce-mail interpreter. This may result in changes in which commands are interpreted. The once harmless commands can be changed to execute unwarranted actions while prohibited commands become acceptable. Conflicts between procedures having the same names may also occur.

• Creation of its own class loader or library loader. This poses serious security implications since it becomes possible to make all commands available to itself including those “dangerous” (such as write, open, delete, etc.) ones. Additionally, this implies that it may be possible to infinitely instantiate from its class and eventually use up all memory space for code and data.

• Termination of its loader. Although this is not a serious security breach, it can cause a lot of inconvenience and potential harm if there are other processes loaded by the same loader. For instance, a break in transaction may affect the integrity of the data being currently processed.

• Denial of service to the recipient's machine. This happens when a ce-mail attempts to interfere with the normal operation of the host system by prohibiting the system to provide any form of service to the user. This can take of form of consuming all the disk space or memory space, or popping up a window so large that the user cannot interact with other programs. From a security viewpoint, denial of service is less severe since they do not cause indelible damages. However, they are most difficult to detect since such an “attack” may not be intentional. The sender of a ce-mail usually does not have prior knowledge of the recipient’s machine and may have designed the ce-mail to deposit a large amount of data.

These aforementioned threats can be broadly classified into three main categories of security attacks, namely, integrity, privacy and denial-of-service attacks.

**Integrity attacks** refers to any form of security breaches that involves changing the state of the recipient machine in an unauthorised manner. An example would be a malicious ce-mail that modifies a data file for the benefit of its designer. This would have changed the state of the system and therefore, constitute as an attack on the system integrity. However, it may be desirable to occasionally permit a trusted ce-mail to have write access to the file system in a limited way.

**Privacy attacks** does not alter the state of the recipient machine, instead, it unlawfully access information about the system without the consent from the recipient. For example, a ce-mail may have access to sensitive information about the recipient and subsequently discloses such information to its designer.

**Denial of service attacks** attempt to deny the user his or her rightful use of the system. Regardless whether such attacks are intentional or otherwise, ce-mails should be prevented from committing such attacks.

In evaluating these security implications, it becomes quickly apparent that it is highly unlikely that any security policy can completely prevent all security attacks. Attempts to eliminate all risks would implicitly inhibit ce-mails to an extent that they are unable to perform any useful function. A reasonably safe ce-mail system should not eliminate all threats but rather is able to reduce them to a manageable level. This is achievable by allowing some form of configurable security for different ce-mails on the basis different levels of trust.
3. Security analysis of the ce-mail system

Figure 1 shows the various components in a ce-mail system. Each component has a distinct role and therefore expected to shoulder different security responsibilities. A ce-mail is first created using a ce-mail authoring tool [2] and transmitted through a ce-mail client. The mail is sent through the Internet via a server (e.g. sendmail server [13]). At the recipient end, the ce-mail is first deposited in a mail repository server and transmitted to the mail client when it connects to collect mails. Subsequently, the ce-mail is loaded and executed by an interpreter.

3.1 Sender client and server

The sender, client or server cannot be relied on to protect a recipient and, hence, should not be given the responsibility to protect the recipient from receiving malicious ce-mails. A ce-mail, regardless of whether it is dangerous or not, must originate from a sender. The sender may not necessarily be the author of a malicious ce-mail but may be victimised to forward one unintentionally. In another situation, the sender client may have been used by unauthorised people posing as rightful owners.

3.2 Recipient server

Unlike the sender server, the recipient server plays a more important role in accepting ce-mails from the Internet. One possible way to step up security is to have firewalls that allow entry to ce-mails from certain locations. In this instance, the security of the ce-mail system is also dependent on the firewall implementation. However, ce-mails coming from trusted servers do not imply that it is safe for execution as in the case of Trojan Horses. Moreover, firewalls can be bypassed through Internet Protocol (IP) spoofing (for example, tagging external ce-mails with internal and trusted IPs).

An alternative is to require the recipient server authenticate the source before accepting ce-mails. However, this may be inefficient or impractical as it will increase network traffic when large amounts of mail are to be handled. Like the above, this solution cannot detect Trojan Horses. On the contrary, this scheme may falsely give recipients the perception that all incoming ce-mails from this process are always safe. At most, the recipient server can alert the users in some ways that the ce-mails may have originated from untrusted sources and hence to exercise extra care.

3.3 Recipient client

With the recipient client being the last interface prior to the execution of the mail, it is therefore not surprising that it has the most crucial security responsibility to shoulder. A number of schemes can
be adopted by the recipient client to enhance the level of security. For example, the ce-mail may be parsed to search for “dangerous” commands at the client machine or the recipient server. Such information can then be fed to the recipient who will make a decision to execute the mail. Ce-mails may be, at the discretion of the recipient, executed locally on a client's machine or moved and executed in a safer environment elsewhere where security is better enforced.

3.4 The transmission network

The transmission network is an independent entity for which the ce-mail system has no control. Therefore, a reliable scheme to protect against a third-party security breach on the network is needed. At present, most network protocols use the Transmission Control Protocol/Internet Protocol (TCP/IP) that transfer data in plain ASCII text. This implies that a third party hacker may hack into the network, retrieve and alter any data packets in the network. For example, a decent ce-mail can be intercepted and tampered while travelling from sender to recipient and subsequently becomes malicious. Various techniques such as encryption and digital signature technology have been introduced to prevent such third party security breaches. Alternatively, hardware protection such as fibre optic cabling instead of the usual coaxial or copper cabling is also possible since fibre optics are more difficult to tap. Nonetheless, this is a very expensive solution and therefore seldom used.

3.5 Ce-mail language and interpreter

If an interpreter cannot interpret any “dangerous” commands then there will never be any malicious ce-mail. However, such a language and interpreter will render the ce-mail quite useless except for a very small number of applications. Therefore, the responsibility of the ce-mail language and interpreter is to provide safe but powerful features for ce-mails. If this can be done then the security of the ce-mail system becomes effective without paralysing the power of applications of ce-mails. The interpreter is a natural choice to provide users with a safe environment to execute the mail. However, there should be a balance between power and security in the language environment. A safe but powerful language and a safe environment for execution of ce-mails are the two key factors of this research to provide a solution to the ce-mail system.

3.6 The operating environment

The operating environment and operating system forms the lowest software level in the ce-mail system. If this has a set of default security features, then the environment for which the ce-mail executes would be safer. However, two main problems arise. Ce-mails are designed to be platform independent for portability reasons. Thus, relying on the operating platform to provide security features is inappropriate as some operating systems (such as DOS) do not exhibit such features. Moreover, when ce-mails arrive at the machine it becomes a part of the system making it difficult for the operating system to keep track of possible Trojan Horses.

4. Safe execution of ce-mails

There are many ways to design a secure ce-mail system. However, the word ‘secure’ is ambiguous as users have different security needs on different machines containing different information. What is acceptable to one user may not be true for another. A good ce-mail environment should encompass a system with a set of security features that can sufficiently protect the user from unacceptable harm. This can take the form of a safe environment or virtual environment for executing ce-mails, support for different security policies and a choice of a suitable ce-mail language.
4.1 Safe or virtual environments for executing ce-mails

Ce-mails should be executed in safe environments where it cannot affect the recipient machine. This may not necessarily take the form of a physical location but a virtual environment such as a virtual machine. A number of techniques can be used to provide such an environment:

**Executing ce-mails at a lower priority.** Ce-mails can be given a priority lower than its recipient. For example, the UNIX environment supports a "nobody" user who has the lowest priority in the whole system. With this, it may be possible to execute the ce-mails so that it cannot perform any malicious acts on the system. Nonetheless, this is not a fool-proof system as different user requirements may dictate the creation of many “nobody” users with different security privileges that make the maintenance of the system difficult. Furthermore, such priority schemes may not exist in other operating platforms such as the Windows 95 platform.

**Creating a separate disk partition for executing ce-mails.** A separate disk partition may be specially created for ce-mail execution so that it will not have access to the file system in another disk partition. Additionally, different operating systems may be installed to execute the ce-mails in this environment. For example, Linux (the personal computer version of UNIX), with its superior security features, may be used in place of Windows 95 for personal computer clients.

**Creating a virtual machine for executing ce-mails.** By using a suitable programming language, a virtual environment (or machine) can be created, destroyed or modified as and when necessary. It will be easier for users to customise a virtual machine based on their security needs than modifying a physical system. Moreover, more than one virtual machine can coexist on the same physical machine independently. A master virtual machine may also have a centralised control over all the slave virtual machines when necessary. Virtual machines are implemented in software and facilitate easy deployment.

Executing ce-mails in such an environment has the advantage of denying the ce-mails of the knowledge of the network information, file systems, and system properties. The ce-mails perceive their environment as a machine in isolation. Depending on the need, special privileges can be granted to some of the virtual machines. The slave virtual machines have priorities lower than that of the master virtual machine. Since the slave machines are controlled independently, they will be ignorant about the existence of other virtual environments and therefore, cannot breach security by communicating or composing their privileges.

Figure 2 shows the virtual environment that can be used to execute ce-mails. The ce-mail client on receiving a ce-mail, invokes a master interpreter (or master virtual machine) and furnishes it with parameters that indicate the security policy to be adopted for that particular ce-mail. A security policy is a set of security privileges for which the ce-mail is granted. This may include access rights for file access, mailing facilities, and so on. The master interpreter subsequently uses this information to invoke a slave interpreter (or slave virtual machine) with the granted privileges to execute the ce-mail. Since the slaves only have the knowledge of the existence of their master, they can only communicate with the master through the virtual communication ports. If two or more slaves need to communicate, they will have to communicate through the master. This provides the master with control over the slaves. The virtual communication ports for every slave may be different depending on the security privileges granted. This concept of a virtual environment has been prototyped by the authors at Nanyang Technological University of Singapore [1].
Virtual communication ports

Master interpreter invokes Slaves with different privileges

Ce-mails executed here

Slave 1

....

Slave N

Figure. 2: Virtual execution environment for ce-mails

4.2 Support for different security policies

Different ce-mails is likely to require different security policies as it is impossible to create a policy that can be used by all applications while maintaining good security. In this context, a security policy is a set of security privileges granted to a particular ce-mail. In defining security policies, it is paramount to be aware that the composition of safe features may not necessarily be true. Therefore, if feature A is safe and feature B is safe, the composition of features A and B is not necessarily safe. For example, it is safe to provide mailing facility to a particular ce-mail. It is also safe to grant read access to a designated directory to another ce-mail. However, if the third ce-mail has access to both of these features then sensitive information in the designated directory can be stolen and sent out through mailing. This is a clear case of a breach in security.

Thus, it becomes important to carefully select the type of privileges to grant to ce-mails. Policies can be classified into internal or external policies [9]. An example of an external policy is the mailing facility while file access is an internal policy. A combination of both policies effectively opens up the file system to the external world. A ce-mail may only use a single security policy in its lifetime. Once it has successfully loaded one policy, it may not load any other policy, even if it has given up on the previous one. Changing the security policy for a ce-mail will effectively compose the features of the security policies, which is not safe.

The safe interpreters (slave virtual machines) have their independent virtual space in the system. They do not possess any knowledge of the existence of other safe interpreters, neither can they communicate directly with each other. This is another essential security feature [9]. If two or more ce-mails can communicate with each other, they can also compose their privileges. For example, one ce-mail that has read access to a designated directory can retrieve a sensitive piece of information and send it to another ce-mail that has only mailing facility. The second ce-mail can subsequently mail the information to the external world, and thus should not be allowed.

5. Safe ce-mail language

The ability to support the concept of a virtual environment largely depends on the ce-mail language used to construct and interpret them. Hence, a suitable ce-mail language is perhaps the one of the most important component in the ce-mail system. If not chosen properly, the language can allow a
seemingly harmless ce-mail to get into the recipient machine and later turn into a malicious Trojan Horse. In contrast, an overly restricted language will render the applications of ce-mails quite useless.

Borenstein has identified four main design issues for a safe ce-mail language in his research of the Enabled Mail Model [3]. These issues are security, power, extensibility and standardisation.

**Security** is concerned with the ability of the language to execute a ce-mail without the ce-mail endangering the recipient or machine any harm. Thus, traditionally powerful languages that do not have control over execution security, such as C and FORTRAN, are completely unsuitable. A suitable ce-mail language need not be entirely new. It can also be a modified language (e.g. Borenstien developed a modified set of Tool Command Language (Tcl) into Safe-Tcl for this purpose [4]). The basic idea is quite simple: the dangerous features of the primitive language should be removed, leaving behind a subset that is safe. The general purpose primitives, which are potentially harmful, are replaced by special but restricted primitives so that useful functions can be performed without jeopardising the security. However, the biggest challenge is to decide which primitives to remove and which primitives to be added to the language. Alternatively, a completely new language can be developed with built-in security features. An example is the Java programming language developed for Internet applications [10][11].

Once a language is made safe, the question of the language’s **power** becomes important. If the language of the ce-mail becomes so restrictive then the areas of application also becomes restricted. A fundamental conflict arises as the potentially dangerous commands are also likely to be the most useful primitives in a programming language. If a language is to be used for a safe ce-mail language without limiting too much of its power, then general purpose commands should be replaced by more specialised and safe ones.

**Extensibility** refers to the ease at which the language can be easily extended to cater for new user requirements and demands. For example, an obvious solution to insufficient power is to add more safe primitives to the existing set. However, this is not an easy task. First, the security aspect of the extension should not be compromised. Security breaches often occur when an existing system is made more convenient. An example is the addition of .rhosts file in the user’s directory. This is supposed to provide convenience by allowing users to remotely login to other machines without the trouble of re-typing the password. This convenience was heavily exploited by the Internet Worm [12]. Assuming that there is no security deficiency resulting from the primitive extension, there remains the practical problem of upgrading all ce-mail systems to conform to the latest version. This leads to the final issue of standardisation.

**Standardisation** and portability are of utmost importance if ce-mails are to be made to operate heterogeneously on a wide range of operating platforms. The sender of a ce-mail has no prior knowledge of the recipient’s machine. Nevertheless, the ce-mail must adopt the same 'look and feel' at the recipient’s machine. A ce-mail can be created in a personal computer under the Windows 95 environment and subsequently executed in the UNIX, OS/2 or Macintosh OS environment. However, in the relentless pursuit of achieving interface portability and standardisation, one may end up with a uniformly bad interface. For instance, the interface quality of a MS-DOS program is never the same or comparable to a Windows 95 program. If one same interface quality is to be achieved across all different platforms, then it is likely to result in nothing more than a text-based question-and-answer ce-mail. Thus, in order to achieve a balance between interface portability and quality, some less popular or outdated operating platforms will have to be excluded.
These aforementioned constraints dictate that a majority of traditional programming languages in their original form do not qualify as a safe ce-mail language. However, if a language can be made safe then it can be a contender as a ce-mail language. Two such languages, Safe-Tcl and Java, are evaluated for their suitability as a ce-mail language. Safe-Tcl is used in a secured ce-mail prototype in [1] while Java is proposed as a safe object oriented language for networked environments.

5.1 Safe-Tcl

Safe-Tcl [9] is basically an extension of Tcl [7] with new features including slave interpreters that are safe by default. The language makes it possible to execute programs without having to know their origin or trustworthiness. It has added mechanisms to control privileges according to their levels of trust. In Safe-Tcl, foreign and untrusted programs are called applets. If the author of an applet can be authenticated, then it can be executed with increased privileges. Two different versions of Safe-Tcl with the same name exist. The developers of the ‘new’ Safe-Tcl from Sun Microsystems drew much inspiration from an original modified and restrictive Tcl of the same name developed by Borenstein [4].

Safe-Tcl is based on the padded cells approach [9]. Untrusted scripts (e.g. ce-mails, applets) are executed under separate environments that are isolated from the trusted segments of the application (e.g. the master interpreter for ce-mails). Its security model offers two key facilities: safe interpreters that provides restricted virtual machines for the execution of applets and aliases which are used by the applets for requesting services from the master interpreter. Unsafe commands are hidden from the safe interpreters. The Safe-Tcl security model does not prescribe any particular security policy except for the most restrictive one, that is, one without any “harmful” primitive. However, mechanisms are provided through aliases to improve the execution power of programs.

![Figure 3: The basic Safe-Tcl mechanism]

The alias mechanism, as shown in Figure. 3, makes it possible to provide restricted access to facilities that are otherwise unsafe, such as file or network access. Different security policies may be implemented by providing different sets of aliases in a safe interpreter. Different organisations can choose to implement different security policies depending on their needs. In particular, it is
possible to implement a default but highly restrictive policy for applets of unknown origin, while less restrictive ones are granted for those whose senders are known or trusted.

Safe-Tcl is an interpreted scripting language. It has a simple syntax that is based on commands consisting of text. Thus, programs created in Safe-Tcl can be executed on any machine as long as there is a valid interpreter on it. To date, there exists Safe-Tcl interpreters on most popular operating platforms such as Windows 95, Windows 3.1, OS/2, MacOS, UNIX, LINUX, and so on. Being an interpreted language, Safe-Tcl programs cannot be infected with virusus.

5.2 Java programming language

Java [10] is a new programming language developed for the Internet. It is a truly interpreted object oriented language. Its programs are first compiled into bytecodes and transferred across the Internet to the recipient machine to be interpreted. Since it is designed as an Internet language, it is secure, provides high performance, highly robust, platform independent and distributed in nature.

Being object oriented, it supports data encapsulation making programs less prone to attacks from other programs. It is designed to operate in distributed environments which imply that security is of paramount importance. The security features incorporated into the language and run-time system allow users to construct applications that are not easily invaded from the outside world. In a networked environment, applications written using Java are protected from intrusion by unauthorised code attempting to by-pass the security and create viruses or invade file systems.

Java scripts require a Java interpreter or a Java virtual machine to execute. The Java virtual machine is an environment provided by the Java interpreter or an operating system with a built-in Java interpreter. Before execution, the Java virtual machine verifies the bytecode to ensure that viruses are not attached to the code. Java programs that are not trusted by the recipient or where source is unknown are called applets. Applets execute within the Java virtual machine rather than directly under the operating platform. Security policies can be easily controlled via the virtual machine. Security features are provided by the Security Manager class in the virtual machine.

Java is a high performance language that supports multi-threading. It achieves superior performance by adopting a scheme where by the interpreter can operate at full speed without having to check the run-time environment. The automatic garbage collector is processed as a low-priority background thread, ensuring the high probability memory availability when required. Java consists of a rich collection of classes defined in its library. This enables programs to be powerful yet compact. Java has gained widespread acceptance since its launch in 1995. Java virtual machines can be found on most computing platforms such as Windows 95, OS/2, MacOS, UNIX, Linux, and so on.

5.3 Comparison of languages

Both Java and Safe-Tcl are safe languages because they do not provide pointers and exhibits good memory allocation. Java suffers a deficiency in its security model in that it provides only a single virtual machine where objects of different classes co-exist. Some objects belong to the untrusted programs (applets) while others belong to the local and trusted program (e.g. applet loader). Since the objects carry different security privileges, this approach makes security handling and privilege allocation rather complex. For example, it can be difficult to differentiate classes belonging to several e-mails having different security privileges and those of the e-mail interpreter.

In Safe-Tcl, the security boundary is simpler and clearly defined. Both data and code of a program or applet exist in only one interpreter and therefore, only one virtual machine. The security policy is
the same for every item in that particular virtual machine. There are no security issues as long as execution stays within the safe interpreter. Security issues only arise when execution goes beyond the safe interpreter through to the master interpreter via the aliases. Thus, security controls can be easily implemented using aliases. The grouping of a set of aliases constitutes a security policy and one applet has only one policy for security reasons described previously.

![Diagram showing Padded Cells and Object Oriented models for security]

Figure 4. Object oriented versus padded cells approach to security

Figure 4 shows the difference between the object oriented approach and padded cells approach to security implementation [9]. The complexity of security issues in the padded cells approach grows with the number of programs (and thus virtual machines). Most of the time, there will only be two, one for the trusted program (master interpreter) and the other for the untrusted applet (safe interpreter). In this case, there is only one security policy employed. In the object oriented approach, the security complexity increases with the number of classes and is related to the functionality of the system. However, it is possible to emulate this padded cells approach in object oriented programming by grouping objects with the same security properties together.

Table 1 summaries the main features between Safe-Tcl and Java languages. It is apparent that Safe-Tcl, Java and other languages have their own advantages and disadvantages. As such, it becomes difficult to advocate a particular language. As long as they meet the requirements specified earlier, they are all possible contenders for implementing a ce-mail system.
Table 1: Features comparison between Safe-Tcl and Java

<table>
<thead>
<tr>
<th>Features</th>
<th>Java</th>
<th>Safe-Tcl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Objected Oriented</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Robust</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Secure</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Interpreted</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Portable</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Exceptions (Built-In)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Multi-Threading</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Safe Environment</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Dynamic Allocation Of Privileges</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Ease Of Configuring Security Privileges</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Ease Of Extending Language</td>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>

6. Conclusions

As opposed to static emails, ce-mails are active mails that are executed upon arrival at the recipient's machine. Their applications lie in areas where static information display is insufficient and where interaction between the recipients and mails can enhance co-operative work. Being potentially powerful, ce-mails can originate from senders with malicious intentions to cause harm to the recipient or recipient's machines. As such, security becomes a paramount factor in the successful design and deployment of ce-mails.

A possible solution to prevent security attacks is to execute ce-mails within a controlled and safe environment. This environment may be a secured operating system, a totally isolated physical machine or a safe but virtual environment that can exist on multiple platforms. Subsequently, ce-mails are executed in safe environments with privileges granted according to their level of trust. The trustworthiness of the ce-mails can be judged by authenticating their developers and senders. However, this is not a fail-safe approach as a best friend can potentially become the worst enemy. Hence, ce-mails should always be given sufficient but restricted privileges.

This paper has highlighted four fundamental requirements for a safe language, namely, security, power, extensibility and portability. Two existing languages, Safe-Tcl and Java, are introduced and evaluated. Both exhibit advantages and disadvantages but are possible contenders for ce-mail systems. Security in ce-mail systems does not rely on one particular security measure but involves preventive measures from all aspects of the system. This can range from the mail authentication through cryptography and digital signatures to the user interface at the recipient's machine. User interface plays a subtle but important role in security. Good user interfaces can promptly alert users of potential dangers in the ce-mail systems but should not put users off by having fanciful but ineffective displays.

References


