INFORMATICA, 2000, Vol. 11, No. 1, 49–64
© 2000 *Institute of Mathematics and Informatics, Vilnius*

Mobile Code Alternatives for Secure Environments

Algirdas PAKŠTAS
University of Sunderland, School of Computing, Engineering and Technology
Chester Road, Sunderland SR1 3SD, England
Institute of Mathematics and Informatics
Akademijos 4, 2600 Vilnius, Lithuania
e-mail: a.pakstas@ieee.org

Igor SHAGAEV
Institute of Control Sciences
Profsoyuznaya St. 65, Moscow, Russia
e-mail: ish@ipu.rssi.ru

Received: March 2000

Abstract. Growing popularity of the mobile code requires to consider various aspects related to its security. In the aviation industry there is a case when additional information needs to be delivered to the pilot by uploading it from the ground station. It creates a need for a platform-independent solution and it raises a problem of the mobile code security as well. Organization of the security in the Base System (similar to extranets) as well as the security issues of the mobile code (or safelets for use in the aircrafts) delivered to the Remote Platform are discussed in the paper. Safelet implementation technologies and issues of code effectiveness and safety itself are discussed with Java and Juice/Oberon technologies been compared.

Key words: mobile code, security, Java, Oberon/Juice, aviation applications, safelets.

1. Introduction

Use of the mobile code becomes more and more popular what is in particular driven by the growth of the Internet and WWW. There are, however, strong concerns expressed by the experts about potential consequences affecting security of the environment which is accepting a mobile code from the third parties. Especially active research and development of these subjects is ongoing in the area of extranets because of their direct relationship with the e-commerce and direct influence on possible financial losses (see, e.g., Baker, 1997; Maier, 1999; Pakštas, 1999; Pakštas, 1999). There are, however, some other areas which also need a special attention to the security of the mobile code, for example area of intelligent transport systems.

Every system using mobile code has a “base system” (BS) and a “remote platform” (RP) to/from which mobile code is sent/received. In a case of ordinary Internet connection the BS can be, for example, service supplier and the RP a service receiver while service is delivered by the mobile code sent from the BS. In a case of transport systems (aviation, ground transport) the RP can be physically mobile and communications are achieved via wireless means. In every case there exist a hazard that both, BS and RP, are attacked using available communications means. Thus, it is a general problem which, however is difficult to solve in simple and universal way. This paper is focused on the organization of the security in the BS (similar to extranets) as well as on the security issues of the mobile code delivered to the RP.

1.1. *Base System Security Issues*

Most often regardless of the business type, sufficient number of users on many of the private networks are demanding access to the Internet services such as the World Wide Web (WWW), Internet mail, Telnet, and File Transfer Protocol (FTP). Additionally, some corporations want to offer WWW home pages and FTP servers for free public access on the Internet. Thus, exposing of the organization's private data and networking infrastructure to the Internet crackers definitely increases concerns of the Network Administrators about security of their networks (Herold and Warigon, 1998). It has been very well expressed that "security should not be a reason for avoiding cyberspace, but any corporation that remains amateurish about security is asking for trouble (Martin, 1996)".

The term security in general refers to techniques for ensuring that data stored in a computer cannot be read or compromised. Obvious security measures involve data encryption and passwords. Data encryption by the definition is the translation of data into a form that is unintelligible without a deciphering mechanism. A password, obviously, is a secret word or phrase that gives a user access to a particular program or system.

Extranet is new type the applications which was introduced relatively recently (Baker, 1997; Maier, 1999). Security issues of the extranets are attracting large publicity both in the technical/research publications (Herold and Warigon, 1998; Lister, 1998) and in the media. In the environments employing mobile code where the BS could be connected to the external network such as the Internet we have a need to look at the BS as a special type of the extranet and apply findings known about extranets to the BS in order to ensure its security.

1.2. *A Need for the Mobile Code in the Aviation Industry*

Safety provision for the vehicles such as aircraft, spacecraft, trains, ships, cars, etc. is of special importance because any serious technical failure could result in catastrophic consequences including loss of human life.

Recent projects developed by EUROCONTROL such as Datalink (ODIAC, 1998) and Human Machine Interface for the Air Traffic Controllers (Eurocontrol, 1998) are addressing mainly on-ground activities as well as ground-to-air *message communication* in a standardized but fairly limited way. The real need, however, exists for systems that will be able to assist the pilot in more active ways, and when failure takes place will either tolerate it or assist graceful degradation to the appropriate (acceptable) state. Moreover, pilots in complex cases are unable to provide correct recovery of the system after fault manifestation.

On-board avionics systems are proprietary and no standardisation is expected in the foreseeable future. The various microprocessors employed dramatically vary in their performance. Motorola 68X2X, PowerPC 603, R4000, and 486DX2 are those to be mentioned here. Thus, in the case where additional information (preferably in easily understandable visual format!) needs to be delivered to the pilot by uploading it from the ground station, there is an obvious need for a platform-independent solution. There are the following two ways to implement it:

- add some additional equipment especially dedicated to receiving this extra information;
- use the same processing facilities which already exist but provide software which will be able to accept some sort of standard document format.

The first method has the advantage that it can be totally isolated from control contours and it can be standardised in the same way as radio communication and navigation systems. However, it requires additional hardware on-board and it must be fitted in the same weight/space limits devoted to the avionics system. Additionally, since the nature of this equipment is to provide assistance to the pilot in special situations, it may never be used at all during the aircraft life. This consideration significantly reduces attractiveness of the first approach. The second method requires the choice of a standard format accepted by all aircraft manufacturers (also true for the first method!) and to implement a software system for every processor on-board to handle uploaded information.

There is one example of this in the computer industry where the PostScript format is used by many printers in order to produce pictures while printers are built using different microprocessors. In this case each printer type is equipped with the PostScript-interpreter.

Thus, functionally such upplodable components will upgrade on-board avionics systems during the flight for achieving better safety. These uploadable components has been named *safelets* (Pakštas and Shagaev, 1999).

Since safelets are sort of computer viruses which are added to the on-board systems, there is always the hazard that they are not authentic but are sent by irresponsible "jokers", or in case of the military aircraft, directly by enemy forces. Even if they could be not exactly viruses designed for crashing systems and destroying the data but stealthy non-viral code, like Netbus and Back Orifice, such mobile code may spy on systems (Defoe, 1999). Therefore there is a need to investigate appropriate technologies for producing secure mobile code for implementation of safelets.

In (Volpano and Smith, 1998) is argued that mobile programming languages should be designed around certain security properties that hold for all well formed programs. This requires a better understanding of the relationship between programming language design and security. Appropriate security properties must be identified.

In a search for a better platform-independent solution for safelet implementation we will discuss two approaches: *interpreter-oriented* and *on-the-fly code-generation runtime system* represented by Java and Juice correspondingly. Comparative evaluation of the methods to address security issues in Java, Safe-Tcl and ActiveX can be found in (Gritzalis and Iliadis, 1998).

1.3. *Organization of the Paper*

The rest of the paper is organized as following. Section 2 discusses security issues of the BS. Sections 3 and 4 are focusing on the safelet implementation technologies such as Java and Juice/Oberon which are compared. Scenarios of the safelet's use in the aviation industry shown in the Section 5. Section 6 presents conclusions.

2. BS Security Issues

A rise in external access to the BS is coming from telecommuters, business associates, customers, or potential customers, each with a unique set of computing and data requirements. Web environment's peculiarities, which include location irrelevance, statelessness, code and user mobility, and stranger-to-stranger communication adds to it additional specifics (Rubin and Geer, 1998). Thus, the solution will not and should not be the same for all (Trolan, 1998). To choose the correct solution, it is important to understand clearly the available alternatives. There can be identified six basic extranet components as well as some specialized and hybrid solutions (Trolan, 1998): (1) access to the external resources; (2) Internet protocol (IP) address filtering; (3) authentication servers; (4) application layer management; (5) proxy servers; and (6) encryption services. Each is sufficient to initiate business communications, but each carries different performance, cost, and security. Namely security (as recent statistics shows (Lister, 1998)), is the main issue preventing organizations from establishing extranets.

In the rest of this Section we will focus on the following issues: risk assessment, development of the security policy, and establishment of the authentication, authorization and encryption.

2.1. *Risk Assessment*

Risk assessment procedure should answer the following typical questions:

- What are organization's most valuable intellectual and network assets?
- Where these assets reside?
- What is the risk if they are subjected to unauthorized access?
- How much damage could be done – can it be estimated in terms of money?
- Which protocols are involved?

2.2. *Security Policy*

To provide the required level of protection, an organization needs a security policy to prevent unauthorized users from accessing resources on the private network and to protect against the unauthorized export of private information. Even if an organization is not connected to the Internet, it may still want to establish an internal security policy to manage user access to portions of the network and protect sensitive or secret information. According to the FBI, 80 percent of all break-ins are internal.

Policy is the allocation, revocation, and management of permission as a network resource to define who gets access to what (Thayer, 1998). Rules and policy should be set by business managers, the Chief Information Officer and a security specialist – someone who understands policy writing and the impact of security decisions. Network managers can define policy for a given resource by creating an entry in access control lists which are two-dimensional tables that map users to resources.

A firewall is an implementation of access rules, which are an articulation of the company's security policy. It is important to make sure that some particular firewall supports all the necessary protocols. If LANs are segmented along departmental lines, firewalls can be set up at the departmental level. However, multiple departments can often share a LAN. In this case, the creation of virtual private network (VPN) for each person is highly advisable.

Following are recognized as a basic steps for developing a security policy:

1. Assessment of the types of risks to the data will help to identify weak spots. After correction, the regular assessments will help to determine the ongoing security of the environment.
2. Identification of the vulnerabilities in the system and possible responses, including operating system vulnerabilities, vulnerabilities via clients and modems, internal vulnerabilities, packet sniffing vulnerabilities and means to test these vulnerabilities. Possible responses include encrypting data and authenticating users via passwords and biometrically.
3. Analysis the needs of user communities:
   - Grouping data in categories and determining access needs. Access rights make the most sense on a project basis.
   - Determining the time of day, day of week and duration of access per individual are the most typical procedures.
     Determination and assignment of the security levels can include the following, five levels:
     - *level one* for top-secret data such as pre-released quarterly financials or a pharmaceutical firm's product formula database;
     - *level two* for highly sensitive data such as the inventory positions at a retailer;
     - *level three* for data covered by non-disclosure agreements such as six month product plans;
     - *level four* for key internal documents such as letter from the CEO to the staff;
     - *level five* for public domain information.

     In order to implement this security hierarchy it is recommended to put firewalls at the personal desktop, workgroup, team, project, application, division, and enterprise level.
4. Writing the policy.
5. Development of a procedure for revisiting the policy as changes are made.
6. Writing an implementation plan.
7. Implementation of the policy.

2.3. *Authentication, Authorization, Encryption*

When it comes to the security aspects of teleworking and remote access, in all aspects of information technology security, there is a tension between the goals of the participants. Users want access to information as quickly and as easily as possible. Whereas information owners want to make sure that users can only access the information they are allowed to. Security professionals often find it difficult to reduce this tension because of the demands of users in a rapidly changing business world. Smartcards may provide the solution for this problem (Baker, 1997).

Involving encryption requires introduction of the key *management/updating procedure*. Encryption can be implemented:

- *At the application*: Examples of this are Pretty Good Privacy (PGP) and Secure Multipurpose Internet Mail Extensions (S/MIME), which provide encryption for e-mail.
- *At the client or host network layer*: The advantage of this approach is that it will provide extra protection for the hosts that will be in place even if there is no firewall or if it is compromised. The other advantage is that it allows distribution of the burden of processing the encryption among the individual hosts involved. This can be done on the client with products such as Netlock (see `http://www.interlink.com/NetLOCK/`), which provides encryption on multiple operating system platforms at the IP level. A system can be set up so that it will only accept encrypted communications with certain hosts. There are similar approaches from Netmanage, and FTP Software.
- *At the firewall network layer*: The advantage to this approach is that there is centralized control of encryption which can be set up based on IP address or port filter. It can cause a processing burden on the firewall, especially if a lot of streams have to be encrypted or decrypted. Many firewalls come with a feature called *virtual private network (VPN)*. VPNs allows encryption to take place as data leaves the firewall. It has to be decrypted at a firewall on the other end before it is sent to the receiving host.
- *At the link level*: The hardware in this case is solely dedicated to the encryption process, thus off-loading the burden from a firewall or router. The other advantage of this method is that the whole stream is encrypted, without even a clue as to the IP addresses of the devices communicating. This can only be used on a *point to point link*, as the IP header would not be intact which would be necessary for routing.

Products like those manufactured by Cylink (see `http://www.cylink.com/products`) can encrypt data after it leaves the firewall or router connected to a WAN link.

Extranet routers combine the functions of a virtual private network (VPN) server, an encryption device and a row address strobe (Roberts, 1998). The benefits of using the extranet routers includes the network's ability to build secure VPNs and tunnel corporate Internet protocol traffic across public networks like the Internet. Management is a lot easier than it is in a multivendor, multidevice setup. Expenses are a whole lot lower, since there is no need for leased lines.

2.4. *Summary of the Weak Points and Security Hazards*

Finally, Table 1 provides a summary of the weak points in the system security, identifies and shows how these problems can be addressed and suggests technical solutions for them. Additional information for various platforms can be obtained in (Castano *et al.*, 1995; Farrow, 1991; NCSA's Publication Catalog; Netware Security; Sutton, Windows NT Security Guide).

Table 1
Weak points and security hazards

Weak Point/Hazard	Technical Solution
Operating system/ applications on servers	Research vulnerabilities; Monitor CERT advisories; Work with vendors; Apply appropriate patches or remove services/applications; Limit access to services on host and firewall; Limit complexity
Viruses	Include rules for importing files on disks and from the Internet in security policy; Use virus scanning on client, servers and at Internet firewall
Modems	Restrict use; Provide secured alternatives when possible (such as a dial-out only modem pool)
Clients	Unix: Same as server issues above; Windows for Workgroups, Win95, NT: Filter TCP/UDP ports 137,138,139 at firewall; Be careful with shared services, use Microsoft's Service Pack for Win95 to fix bugs
Network snooping	Use encryption; Isolate networks with switch or router
Network attacks	Internet firewall; Internal firewall or router; Simple router filters that don't have an impact on performance
Network spoofing	Filter out at router or firewall

3. Java

3.1. *Why Java?*

Let's look at the three main reasons which make Java potentially suitable for safelet implementation.

First Reason: It is a programming language (i.e., *standard format*) which was specially designed for creation of platform-independent software. It works in such a way that every computer which wants to run Java-code, must be equipped with the *interpreter* and support libraries to implement necessary calculations or visualisation tasks.

Second Reason: Java has proved to be a feasible solution in the heterogenous computing environments *with dynamic uploading* of the information according to the user requirements.

Third Reason: Java applets are very much suitable for implementation of *the visualisation and animation* tasks which are important for time-constrained situations.

Java's distribution format is a sequence of so-called byte-codes. The inventors of Java thought up an "ideal" processor designed specifically for running Java programs and used the instruction set of this ideal processor as the "intermediate language" in which Java programs are shipped. It is entirely conceivable that the ideal Java processor can actually be implemented in hardware, and Sun has actually already announced such processors.

To summarise, Java includes the following features (Sun Microsystems, 1995): object-oriented language with data-centric model, native multi-threading support, garbage collection, no pointers, no macro support, compiler-generated symbolic byte code (interpretable), restricted access to run-time environment, runtime verification of code integrity, digital signature, encryption.

3.2. *Critics against Java*

Let's now discuss the arguments against Java.

Java is not real-time: This is the most obvious argument to reject Java for implementation of the mission-critical software. This argument, however, is based on the presumption that all avionics software must be hard real-time software. This is, in our opinion, not very applicable to safelets because most likely they will not be directly used to control aircraft but rather will try to assist the pilot by providing additional information. Finally, if it is still needed, there is already a published proposal (Nilsen, 1996) for a real-time dialect of Java which was enthusiastically evaluated by the developers of both kinds of applications – embedded systems and ordinary Java applications. Thus, we can expect a stable version of the real-time Java fairly soon.

Java applications are slow: This issue is related to the previous one and it is pretty much true because of the nature of the Java Byte Code interpretation. But in the majority of situations it should not cause a problem because uploadable information does not always have to be animated. Possible scenarios are discussed below.

Java is not safe: A number of security flaws, discovered in early implementations of Java, received a lot of publicity (Dean *et al.*, 1996). Needless to say these errors were corrected as soon as they were discovered.

The *Byte Code Verifier*, *Class Loader* and *Security Manager* has been discussed (Niinimaki *et al.*, 1998) as the means to ensure Java applet security. Applying the verifier requires data-flow analysis and a partial repetition of the compiler's integrity checks. For each machine instruction, it must examine all possible paths leading to the instruction to ensure that the registers hold values of the appropriate type. This analysis requires structural information about the program that is not immediately available in a sequential virtual-machine instruction stream. Extracting the required structural information from such a "flat" representation is a time-consuming task whose complexity grows faster than linear program size. Thus, this feature indirectly makes Java less attractive for implementation of the critical real-time safelets.

There is also the possibility to sign applets with the user's digital signature (Niinimaki *et al.*, 1998). Cryptography functions added to the Java Developer Kit are not sufficient for use worldwide because the U.S. federal government does not allow strong encryption tools to be included in software exported or used outside of the country.

Despart of many efforts to produce more secure Java system it still has been reported (see Intelligent-Enterprise, 1999) that in March 1999, Java 2 fell prey to an attack discovered by a German graduate student.

Thus, the question is whether Java-safelets are secure enough to be used in serious applications. For aircraft with high security demands (such as military), Java is not recommended because the trade-off between the security aspects and its power is not good enough. But in aircraft where security is not a key issue, Java offers benefits along with acceptable security risks. Java applets are still a much better alternative than Microsoft's ActiveX, which does not include sufficient security mechanisms.

4. Juice

4.1. *Origins of Juice*

Juice (Franz *et al.*, 1996) is a new technology for distributing executable code across the World Wide Web and in this respect, it is similar to Java. However, Juice differs from Java in several important aspects that allows it to outperform Java in many applications.

Juice grew out of a research project headed by Niklaus Wirth. His aim was to invent a language that was at once *simple* and *safe to use*. The new language should also support the *object-oriented programming paradigm*, be *efficiently compilable*, and applicable to a wide range of programming tasks from systems programming to teaching. A language with these properties was defined by Wirth in 1988 and given the name Oberon (Wirth, 1988). The overview of the Oberon implementation can be found in (Bradis *et al.*, 1995).

4.2. *Oberon vs. Java*

Oberon looks a lot like Pascal, and Java looks a lot like C. A closer comparison of Oberon and Java shows, however, many similarities:

- both languages are strongly typed while permitting the definition of "extended" types that are backward-compatible with their ancestors,
- both prohibit pointer arithmetic and mandate the presence of a garbage collector in their run-time environment, and
- both provide the notion of modular "packages" that are compiled separately and dynamically linked at run-time.

The type paradigm applied by Java and Oberon has important consequence for the type-safety of the applets and will be discussed in more detail below. Here we just note that both languages are fully type-safe, and hence both are equally suitable for applet-programming.

Juice is to the Oberon language what the *Java Virtual Machine* is to the Java language: the means of distributing portable software.

Juice encompasses three key components:

- an architecture-neutral software distribution format,
- a compiler that translates from Oberon into the Juice portable format, and
- a plug-in that enables Netscape and Microsoft Internet Explorer browsers to run Juice applets.

Thus, Juice has a very similar application domain as Java and shares with the Java the same reasons (see above) which make it potentially suitable for safelet implementation.

4.3. *Juice Outperforms Java*

The most notable difference between comparable Juice and Java applets is probably that the Juice version of any applet is likely to run a lot faster than its Java counterpart (Franz and Kistler, 1996). Rather than being interpreted, as Java applets normally are, Juice always compiles each applet into the native code of the target machine before it begins execution (Franz, 1994). Once an applet begins executing, it runs at the full speed of compiled code. Juice's on-the-fly compilation process is not only exceptionally fast, but it also generates object code that is comparable in quality to commercial C compilers. In contrast, even the latest just-in-time compilers for Java that have been announced by major vendors are not yet trying to compete with regular compilers.

4.4. *Juice Compressed Tree Code vs. Java Byte-Code*

The effectiveness of Juice largely depends on its software distribution format (Franz and Kistler, 1996). This distribution format (called *slim binaries)* is far more complex than Java byte code. It is based on a tree-shaped program representation as is typically used transiently within optimizing compilers. Rather than containing a linear code-sequence that can be interpreted byte-by-byte, a Juice-encoded applet contains a *compressed tree* that describes the actions of the original program (Franz and Kistler, 1996; Franz, 1997). The tree preserves the control-flow structure of the original program, which makes it much easier to perform code optimization while the tree is translated into the native instruction set of the target machine.

In the actual Juice implementation, tree compression and linearization are performed concurrently, using a variant of the classic LZW data-compression algorithm (Welch, 1984). Unlike the general-purpose compression technique described by Welch, however, the algorithm applied in Juice is able to exploit domain knowledge about the internal structure of the syntax tree being compressed. Consequently, it is able to achieve much higher information densities (Fig. 1).

It appears that there are no conventional data-compression algorithms, regardless of whether applied to source code or to object code (for any architecture, including the Java

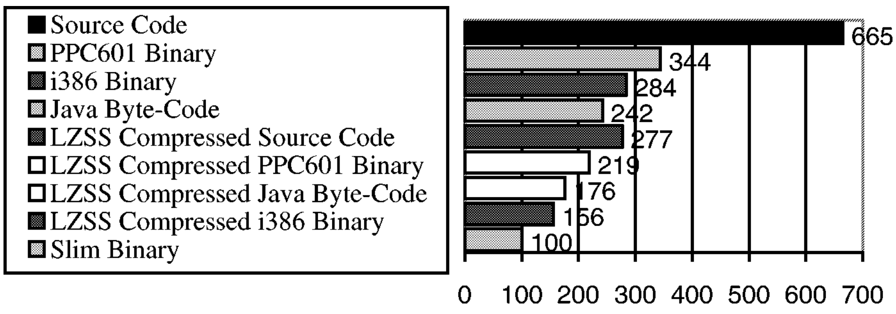

Fig. 1. Relative size of a representative program suite in various formats.

virtual machine), that can yield a program representation as dense as the slim binary format (Franz, 1997).

Thus, users with slow (e.g., wireless) connections (or in emergency flight situations) save time by transmission Juice applets rather than Java applets of the same functionality.

In fact, the time saved by the faster downloading normally more than compensates for the on-the-fly compilation phase.

4.5. *Juice's Enhanced Security*

As it is noted in (Franz, 1997) the topic of mobile-code security is independent of the choice of programming language that a mobile program is written in, and also independent of the representation that is used in transporting applets to target machines. A minimum requirement is that *type-safety* is maintained.

However, absolute type-safety requires more than just compile-time checking because an attacker could hand-craft a malicious applet directly in the mobile code representation without passing it through a compiler, thereby circumventing the type-safety of the source language. As a consequence, mobile code needs to be scanned and verified *prior to execution* even if it is based on a "safe" language (Yellin, 1995).

Juice tree-based encoding avoids many of the security issues that are difficult to solve in any environment based on a virtual-machine similar to Java (Franz, 1994). By the very definition of the tree-encoding scheme, it is impossible to encode a tree that violates scoping rules of the source language. Such an "illegal" program cannot even be constructed by hand. Java's byte-code instructions, on the other hand, are at a much lower semantic level.

As discussed earlier, most of the existing implementations of Java provide code verifier for incoming mobile code (Niinimaki *et al.*, 1998). Juice currently does not includes a code verifier. However, as it is stated by the Juice authors (Franz, 1994) this is simply a restriction of the current implementation of the Juice system, and does not mean that Juice's intermediate representation is less well suited for verification than Java's.

The tree-based Juice distribution format in effect makes this particular data-flow analysis unnecessary. It also allows for highly efficient load-time integrity checking, as every node in the encoded tree is fully typed.

The advantages of Juice are even more obvious for the large applets. Java, on the other hand, has an advantage in the area for which it was originally invented: embedded applications in consumer electronics and household appliances.

5. Scenarios for Using of Uploadable Information

As discussed earlier, human operators can be assisted with the help of additional information provided from the base. Let us look at the possible scenarios of using uploadable information.

5.1. *Aircraft Emergency Landing*

One obvious situation when such additional information can be extremely important is aircraft emergency landing. Civil aviation pilots are normally provided with maps and other important navigation sources but with certain limits imposed by the vehicle purpose (weight, space, etc.). Also, since nobody can be sure where and when an emergency landing will occur, even the most precise navigation sources, while helpful during the ordinary flight can be outdated or not detailed enough in the particular place where the landing must be fulfilled.

Examples of such additional information are location and other important parameters of the new bridges (possible obstacles), highways (possible landing lane), tower control information (avoiding conflicts with other aircraft in the sector).

Such information delivered by the safelets can be either static (e.g., drawings or maps) or dynamic (updatable map of the flights in the segment).

By all means this information must be much more clear and easy to understand and operate than current simple radar displays Operation procedures there are based on manipulating the paper strips with essential flight information.

5.2. *Integration of Ground and On-board Air Traffic Control (ATC) Systems*

There are already efforts to create a universal co-ordination system allowing transfer of aircraft Centre to Centre and sector to sector by means of predefined messages (SYSCO). However, this system is addressing only activities of control towers but is not directly intended to assist pilots. Work to create a paper-strip-less Human Machine Interface (HMI) for the next generation of ATC Systems is under way (EUROCONTROL, 1998).

This HMI can easily be integrated within CoDySa in order to assist the pilot and provide him/her information which the air traffic controller will see on his/her display.

The most attractive feature of this approach is that the HMI prototype is implemented in Java and this fact alone is a step towards a definition of the structure and contents, as well as the development of the standard class libraries for visualisation of the air traffic information. This experience could be useful in the Juice-based approach.

6. Conclusions

On-board avionics systems are examples of environments with especially high requirements for the security operation. They are proprietary and no standardization is expected in the foreseeable future. Thus, in a case when safety enhancing component (safelet) has to be delivered to the pilot by uploading it from the ground station, there is an obvious need for a platform-independent solution and it raises a problem of the mobile code and its security.

The paper analyzed various aspects related to both base system (BS) and remote platform (RP) security and importance of the employing an adequate technology for producing the mobile code has been emphasized. From the comparison of the Java and Oberon/Juice we conclude that the latter is more suitable solution for the secure environments.

There are following problems which should be solved before introducing safelets:

- Introducing regulations for the types of information uploadable to the aircraft. Set of messages defined in the HMI project (EATCHIP PHASE III Human Machine Interface) gives us a clear guideline for the selection of the appropriate information.
- Clarification of the status of the safelet in the process of making the decision by the pilot (information only, advice, strong advice, direct instruction from the control tower, etc.).
- Security mechanism for authentication of the safelets should be built in order to avoid hazards of the Trojan Horse.
- Finally, building infrastructure for uploading of safelets. This can be done gradually starting from pilot-project zones and extending experience worldwide.

Implementation of safelets (both for real-time and not) will need development of:

- industry-standard, reusable and extendable software component library (classes);
- development tools for various purposes such as human-machine interface generators, validation tools for Datalink messages, etc;
- debugging and monitoring tools.

References

Baker, R.H. (1997). *Extranets: The Complete Sourcebook*, McGraw-Hill Book Company.

Birch, D. (1998). Smart solutions for Net security. *Telecommunications*, **32**(4), 53–54.

Brandis, M., R. Crelier, M. Franz and J. Templ (1995). The oberon system family. *Software – Practice and Experience*, **25**(12), 1331–1366.

Castano, S, *et al.* (1995). *Database Security*, Addison Wesley Publishing Co. – ACM Press.

Dean, D., E.W. Felten and D.S. Wallach (1996). Java security: from HotJava to netscape and beyond. In: *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, Oakland, California, pp. 190–200.

Defoe, D. (1999). Malware: the universe of destructive code, *Secure Computing*, 28–30, 32, 34, 36, 38.

EATCHIP PHASE III Human Machine Interface (HMI). EUROCONTROL (1998). (`http://www.eurocontrol.be/projects/eatchip/hmi/`).

Farrow, R, (1991). *Unix Systems Security*, Addison-Wesley Publishing Co.

Franz, M. (1994). *Code-Generation On-the-Fly: A Key to Portable Software*. Zürich: Verlag der Fachvereine.

Franz, M. (1997). Adaptive compression of syntax trees and iterative dynamic code optimization: two basic technologies for mobile-object systems. In J. Vitek and C. Tschudin (Eds.), *Mobile Object Systems: Towards the Programmable Internet*, Springer Lecture Notes in Computer Science, **1222**, pp. 263–276.

Franz, M. (1998). Open standards beyond Java: on the future of mobile code for the internet. *Journal of Universal Computer Science*, **4**(5), 521–532.

Franz, M., T. Kistler (1996). Introducing Juice. Irvine: University of California `http://caesar.ics.uci.edu/ juice/index.html`

Franz, M., T. Kistler (1997). Slim Binaries. *Communications of the ACM*, **40**(12), 87–94.

Global Network Security Products, Cylink Corporation (1998). `http://www.cylink.com/products`

Gritzalis, S., J. Iliadis (1998). Addressing security issues in programming languages for mobile code. In *Proceedings Ninth International Workshop on Database and Expert Systems Applications. IEEE Comput. Soc.*, Los Alamitos, CA, USA, pp. 288–293.

Herold, R., S. Warigon (1998). Extranet audit and security. *Computer Security Journal*, **14**(1), 35–40.

(1999). Java 2: the real security risks. *Intelligent-Enterprise*, **2**(9), 34–3, 38–40.

Lister, T. (1998). Ten commandments for converting your intranet into a secure extranet. *UNIX Review's: Performance Computing*, **16**(8), 37–39.

Maier, P.Q. (1999). Implementing and supporting extranets. *Information-Systems-Security*, **7**(4), 52–59.

Martin, J. (1996). *Cybercorp: the New Business Revolution*, Amacom Book Division.

NCSA's Publication Catalog, NCSA (1998). `http://www.ncsa.com/catalog/catgenerity.html`

Netlock Version 1.4.1, Interlink Computer Sciences, Inc. (1998). `http://www.interlink.com/NetLOCK/`

Netware Security, ALC Press, `http://alcpress.com/`

Niinimaki, P., P. Markkanen, J. Kajava (1998). Java applets and security. In *Databases and Information Systems, Proc. 3rd IEEE International Baltic Workshop*, Riga, Latvia, **2**, pp. 125–136.

Nilsen, K. (1996). Real-Time Java (v.1.1) Iowa State University, Ames, Iowa www.newmonics.com

ODIAC – Operational Requirements for Air Traffic Management (ATM) Air/Ground Data Communications Services, EUROCONTROL (1998). `http://www.eurocontrol.be/projects/eatchip/odiac/`

Pakštas, A. (1999). Towards electronic commerce via science park multi-extranets. *Computer Communications*, 1351–1363.

Pakštas, A. (1999). Security aspects and cost models of the X-tranet systems. *ISAS'99/SCI'99: World Multiconference on Information Systems, Analysis and Synthesis/Systemics, Cybernetics and Informatics*, Orlando, FL, USA, Vol. IV, pp. 526–533.

Pakštas, A., I. Shagaev (1999). Extendable ground-to-air communications architecture for CoDySa. In M. Felici, K. Kanoun, A. Pasquini (Eds.), *Computer Safety, Reliability and Security*, Springer-Verlag, LNCS 1698, pp. 187–201.

Roberts, E. (1998). Extranet routers: The security and savings are out there. *Data Communications*, **27**(12), 9.

Rubin, A.D., D.E. Geer (1998). A survey of Web security. *Computer*, **31**(9), 34–41.

Sutton, S.A. *Windows NT Security Guide*, Addison Wesley Developers Press, `http://www.trustedsystems.com/NTBook.htm`

Thayer, R. (1998). Network security: Locking in to policy. *Data Communications*, **27**(4), 77–80.

The Java language: a white paper. Sun Microsystems (1995) `http://java.sun.com/`

Trolan, S. (1998). Extranet security: what's right for the business? *Information-Systems-Security*, **7**(1), 47–56.

Yellin, F. (1995). Low level security in Java. In *Fourth International World Wide Web Conference, World Wide Web Consortium*, Boston, Massachusetts, `http://www.w3.org/pub/Conferences/WWW4/Papers/197`

Volpano, D., G. Smith (1998). Language issues in mobile program security. In G.Vigna (ed.), *Mobile Agents and Security*, Springer-Verlag, Berlin, Germany, pp. 25–43.

Wirth, N. (1988). The programming language Oberon. *Software – Practice and Experience*, **18**(7), 671–691.

Welch, T.A. (1984). A Technique for high-performance data compression. *IEEE Computer*, **17**(6), 8–19.

A. Pakštas received the M.S. degree in radiophysics and electronics from the Irkutsk State University in 1980 and Ph.D. degree in systems programming from the Institute of Control Sciences in 1987. Professor title has been awarded to him by the Agder College in 1997. Since 1998 he has been with the School of Computing, Engineering and Technology, University of Sunderland. His research interests include software engineering for distributed computer systems, communications engineering and real-time systems. He is elected officer in the IEEE Communications Society Technical Committees on Communications Software (Secretary), Enterprise Networking and Applications (Vice-Chair) and Multimedia Communications (Vice-Chair). Has published 2 research monographs and over 100 other publications. He is a senior member of the IEEE and a member of the ACM and the New York Academy of Sciences.

I. Shagaev received the M.S. degree in electronic and electromechanical engineering from the Moscow Power Engineering Institute in 1977 and Ph.D. degree in fault tolerant computing from the Institute of Control Sciences in 1987. Currently he is a head of Fault Tolerant Computer Branch at the Institute of Control Sciences. His research interests include various aspects of the fault-tolerant computing, computer architectures, hardware and software design and on-board avionics systems. Has published research monograph on these topics and over 50 other publications. He is a fellow of Institution of Analyst and Programmers, UK, 1990.

Mobilaus kodo alternatyvos saugioms aplinkoms

Algirdas PAKŠTAS, Igor SHAGAEV

Augantis mobilaus kodo panaudojimo populiarumas reikalauja atsižvelgti į jo saugumo aspektus. Aviacijos industrijoje pasitaiko atvejų, kada papildoma informacija turi būti perduota pilotui skrydžio metu iš antžeminės stoties. Tai reikalauja sprendimų, nepriklausančių nuo kompiuterio platformos (sistemos), ir taip pat iškelia mobilaus kodo saugumo problemą. Straipsnyje yra nagrinėjami Antžeminės Bazės saugumo organizavimas (traktuojant ją panašiai kaip extranet) bei mobilaus kodo (vadinamo "safeletu") saugumo aspektai, tokį kodą perduodant į Nuotolio Platformą. "Safeletų" kūrimo technologijos bei kodo efektyvumo ir saugumo aspektai diskutuojami, palyginant Java ir Juice/Oberon.