INTERNET ARCHIVE WayBackMachine http://kimera.cs.washington.edu/overview.html Go DEC JAN 15 1998 JAN 1997 1999 About this capture
28 captures
15 Jan 1998 - 7 Jan 2006

Kimera Architecture

Introduction

Executable content on the Internet has flourished through Java. Java enables mobile programs, called applets, to be shipped from a web server to a client's machine, where they execute locally and are integrated into the client's environment. The key technology that Java offers is the ability to verify that an applet is well-behaved. Using Java, a client can ensure that the shipped code will not crash her machine or perform operations for which it does not have permission, such as read private files, erase a disk or send credit card information to third parties. This assurance rests on information contained in Java bytecode, and can be independently verified by any client. It is this ability to independently verify the safety of shipped code that is largely responsible for Java's success on the Internet.

Despite Java's promise of safe mobile code, the current Java security model and the current implementations of Java security verification suffer from a number of serious deficiencies. The current security model necessitates that every client perform its own safety checking. The verifier and JIT compiler/interpreter are located in the same physical machine as the Java Virtual Machine. As a result, the trusted computing base (TCB), that is, the set of software in which one must have inherent trust, is very large. The trusted computing base for an enterprise that uses the current Java implementations is on the order of verifier, compiler and interpreter size times the number of instances of Java virtual machines, such as Java enabled browsers.

The size and distribution of the trusted computing base leads to a variety of security problems. First of all, security verification and access control become discretionary with the current Java implementations. Since the verification process is under the control of end-users, any individual within an organization may disable some or all of the checks performed on imported code. Even in the absence of any malice within an organization, the difficulty of security upgrades may allow outdated and buggy copies of Java verifiers to exist. For instance, our organization has at least eight different versions of the Netscape browser, including some which have been shown to contain security flaws. Furthermore, the current structure of Java based systems makes security management at the level of an enterprise almost impossible. Since access control decisions are at the discretion of users, there is no way to impose a coherent, enterprise-wide security policy. Finally, the high computation and memory requirements of a verifier, JIT compiler and interpreter restrict the applicability of Java technologies to fast, hence costly, clients. Ubiquitous Java code cannot become a reality until the Java virtual machine can run in resource limited consumer electronics.

We suggest an alternative way of structuring Java systems, in which verification and compilation are performed in centralized servers. A centralized security architecture is easy to administer, provides enterprise-wide security, supports safe auditing, and reduces Java client resource requirements. The trusted computing base is minimized under this scheme, and consists of small and simple components whose security can be more readily assured. Consequently, under our architecture:

- security is manageable, centralized, and security restrictions are mandatory, not discretionary
- auditing is performed independently of Java end-points, and audit trails are physically protected from foreign code
- memory and processing requirements at the end nodes are reduced through way-ahead-of-time compilation
- security upgrades and bug fixes do not require end-user assistance or action

Our overall goal is to create a secure, extensible, manageable and high-performance distributed computing infrastructure.

Motivation

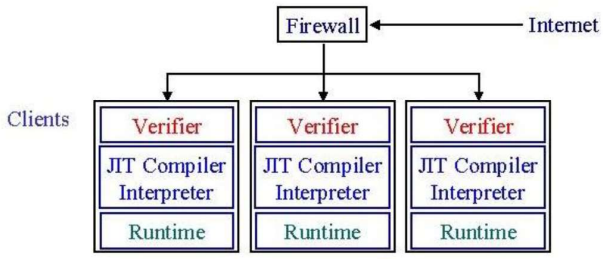
Under the current state of the art Java implementations, each client is individually responsible for correctly validating, auditing, and executing foreign code. This point-to-point approach to distributed system construction is not only open to security attacks, but also ineffective and inefficient.

First of all, security is discretionary under such a point-to-point model. An individual Java end-node can explicitly disable verification or be implicitly compromised by malicious foreign code. Once verification is compromised, a Java end-node becomes a beachhead for further security attacks.

Similarly, a point-to-point model does not lend itself to centralized, enterprise-wide security management. Currently, there are no interfaces (besides disabling Java altogether) by which a system administrator can limit or arbitrate the extent of Java usage among clients within a security domain. Clearly, being able to express and enforce organization-wide restrictions is a crucial feature (e.g. applets should be allowed to touch a sandboxed part of the filesystem, but access to sensitive information should be carefully guarded). As long as access restrictions are under the discretion of users, enforcement of such access controls is optional and error prone.

Further, fixing security problems is very costly under a point-to-point security model. Upgrading JVM implementations at every end-node in response to the discovery of security flaws is a difficult and expensive undertaking. Similarly, keeping secure audit trails is difficult under a point-to-point model, since the first thing malicious code would do after compromising a system would be to destroy the audit trails.

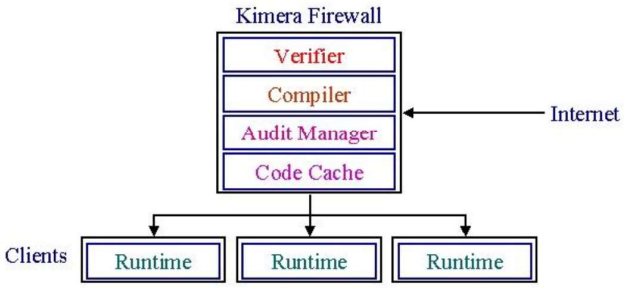
Finally, the point-to-point security model forces every client to support a local verifier and compiler. Hence, the resource requirements for Java end points are very high. Each client has to provide its own verifier, interpreter and optionally a just-in-time compiler. The resource requirements for a Java client are consequently very high. The Java component of the Netscape browser on DEC alphas accounts for about 700K of static code and data, and makes Java unsuitable for small mobile devices.



Current Java architectures are point-to-point, leaving crucial system services up to the discretion of clients.

Approach

Our approach to Java system structure is to break the trusted computing base into small, secure components and locate them in security firewalls. In particular, locating verification, access control, auditing, and compilation services in centralized servers enables the same services to be performed faster, cheaper, and without introducing security vulnerabilities.



Kimera Firewall Architecture

The verification service performs checking of incoming code for typesafety and conformance to interfaces. Moving this service to a centralized server, and reducing it to a single, small component greatly simplifies security management. Since the firewall itself does not run foreign code and exports a simple interface on the network, its security can be more readily ensured than the endpoints. Further, a security upgrade affects only the central firewall, and the change does not require any end-user cooperation. Such a design enables timely security upgrades, especially for small, mobile clients where the Java engine may be hard-coded in ROM.

Locating the verification service at a separate server requires that the code verified at the central service remain intact after verification. Tamper-proof signatures, using a cryptographically strong hash function such as MD5, can be utilized to ensure the safety of the code after it leaves the verifier. A client that trusts the verification service would then need only implement MD5 instead of a full-blown verifier. This makes a sizable reduction in computation and memory requirements of the end-points.

Compilation from Java bytecode into native machine code is another service that is best performed at a central location. Since the compiler is part of the trusted computing base, wherein a single error may result in violation of the security guarantees of Java, it should be protected from foreign code. The same arguments for ease of upgrades and uniformity of enforcement that apply to the verification service apply equally well to the compilation service. However, when the compilation service is moved into a centralized server, it needs to be qualitatively different. In particular, techniques such as just-in-time compilation are not suitable, because the communication overhead between the client and the compilation server would dominate any potential benefits. Instead, compiling incoming Java code ahead of time would reduce communication requirements and enable foreign code to run natively. Such an ahead of time compiler could deliver high performance, as it is financially viable to locate the shared firewall service on a machine faster than any endpoint. Caching the results of recent compilations in the central compilation server can be used to further reduce the latency of binary translation. Without the need for a JIT compiler, verifier, or interpreter, a Java end-point would consist of only a minimal runtime environment. Java enabled network computers could then be constructed from cheaper components.

Further, audit trails may be kept at the secure firewall, where they would be safe from any tampering by malicious code. Since the firewall itself does not execute any foreign code, a bug in the verifier that allowed a rogue application to execute on an end-point would not put the audit trails at risk. The audit trail may then be used to prohibit further entry of such code until a security patch can be applied.

Finally, transparent modifications of the Java bytecode can be performed at the firewall in order to perform fine-grained security checks. For instance, the centralized security policy may allow foreign code to touch only certain portions of the filesystem. The firewall could inject some Java bytecode into every downloaded class to check the arguments to the filesystem open routine to see if that particular instance of the call should be allowed. Hence, rewriting technology at the firewall can be used to implement fine-grain access control.

Results and Status

We have implemented a clean-room Java verifier to serve as the backbone of our firewall. While it is hard to quantify the security of a piece of software, we believe that making the verifier small, simple, and standalone enabled us to implement a more robust verifier than found in commercial systems. Our testing methodology helped strengthen our verifier and uncover numerous flaws in commercial verifiers.

We have implemented a Java disassembler to aid in Java development. The disassembly service is freely available, and generates Jasmin-compatible output.

We are currently implementing the server infrastructure to support centralized compilation and verification.

Emin Gün Sirer & *Sean McDirmid* & *Brian Bershad*

Project Kimera

Department of Computer Science and Engineering
© 1997, University of Washington