US005983348A

United States Patent [19]

Ji

[11] Patent Number: 5,983,348

[45] Date of Patent: Nov. 9, 1999

[54] **COMPUTER NETWORK MALICIOUS CODE SCANNER**

[75] Inventor: **Shuang Ji**, Santa Clara, Calif.

[73] Assignee: **Trend Micro Incorporated**, Cupertino, Calif.

[21] Appl. No.: **08/926,619**

[22] Filed: **Sep. 10, 1997**

[51] **Int. Cl.**[6] **G06F 13/00**

[52] **U.S. Cl.** **713/200**; 714/38

[58] **Field of Search** 395/186, 187.01, 395/188.01, 183.14, 183.13, 200.54, 200.55, 200.32; 380/3, 4, 23, 25; 713/200, 201, 202; 714/38, 37

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,257,381	10/1993	Cook	395/700
5,359,659	10/1994	Rosenthal	380/4
5,390,232	2/1995	Freeman et al.	379/15
5,623,600	4/1997	Ji et al.	395/187.01
5,805,829	9/1998	Cohen et al.	395/200.32

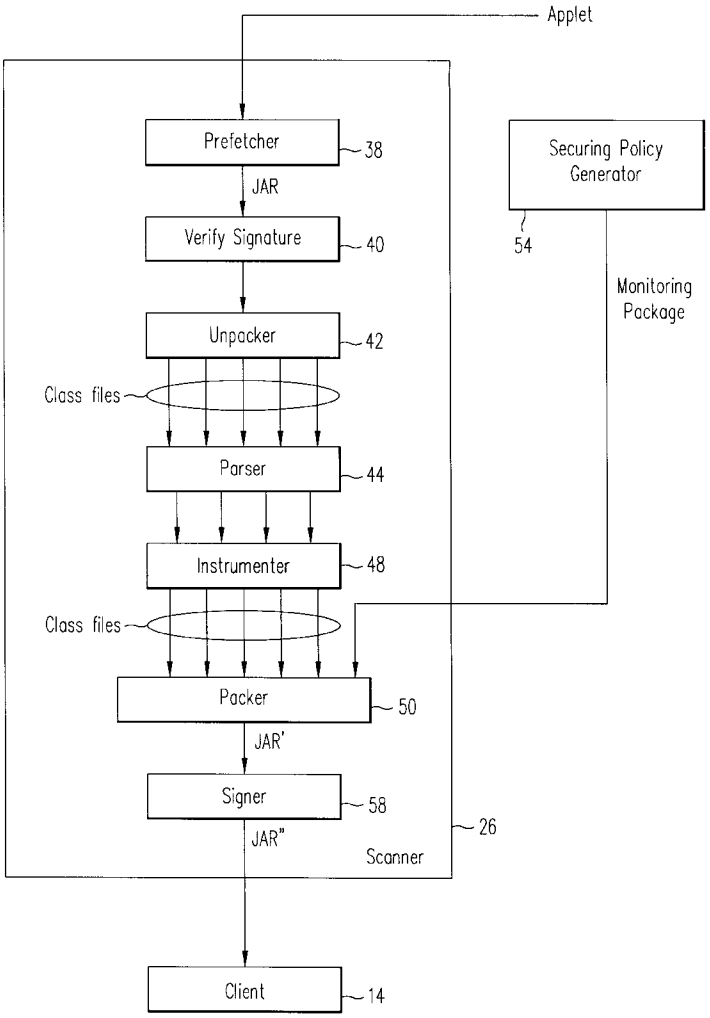
Primary Examiner—Norman Michael Wright

Attorney, Agent, or Firm—Skjerven, Morrill MacPherson, Franklin & Friel LLP; Norman R. Klivans

[57] **ABSTRACT**

A network scanner for security checking of application programs (e.g. Java applets or Active X controls) received over the Internet or an Intranet has both static (pre-run time) and dynamic (run time) scanning. Static scanning at the HTTP proxy server identifies suspicious instructions and instruments them e.g. a pre-and-post filter instruction sequence or otherwise. The instrumented applet is then transferred to the client (web browser) together with security monitoring code. During run time at the client, the instrumented instructions are thereby monitored for security policy violations, and execution of an instruction is prevented in the event of such a violation.

34 Claims, 2 Drawing Sheets



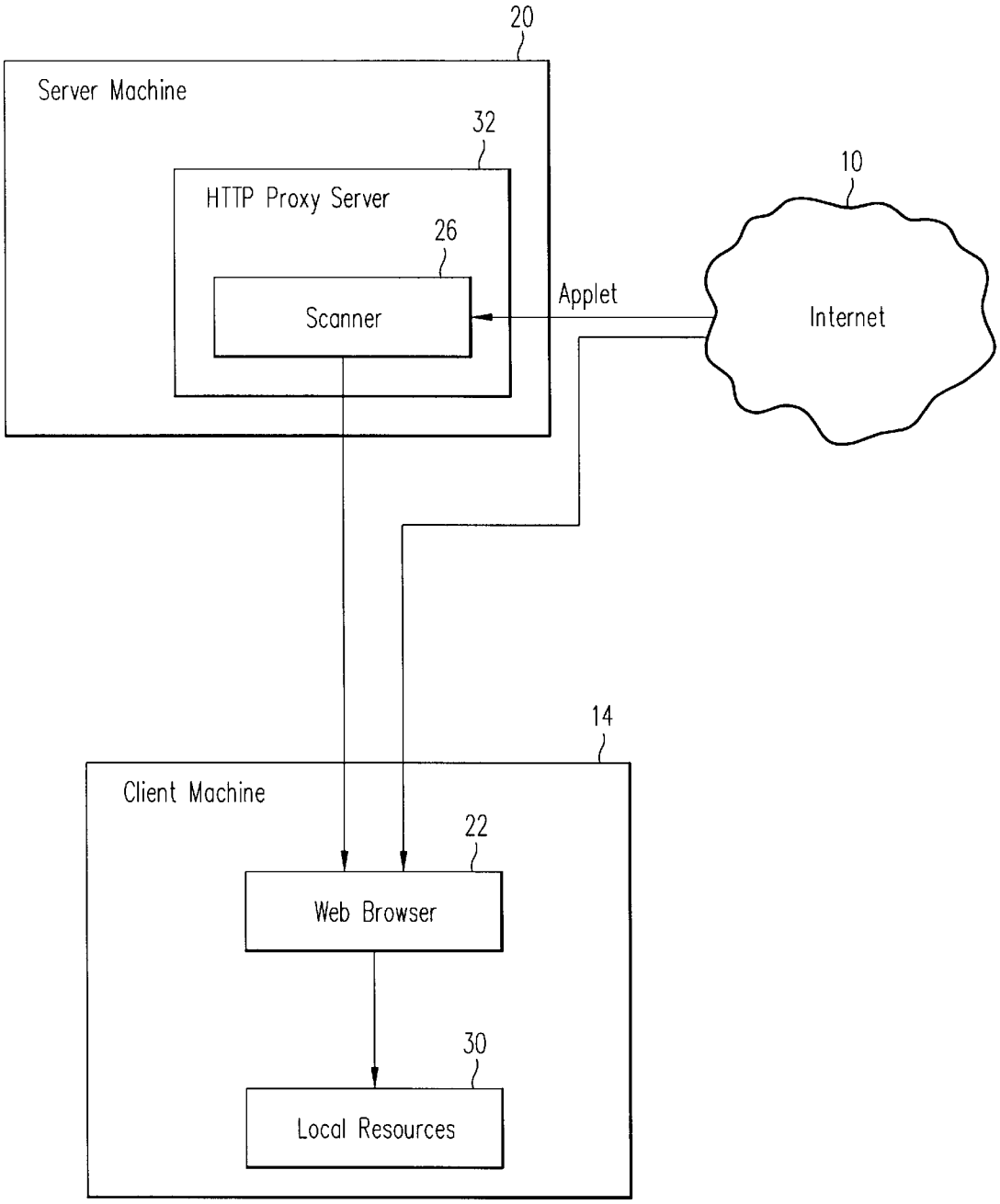


FIG. 1

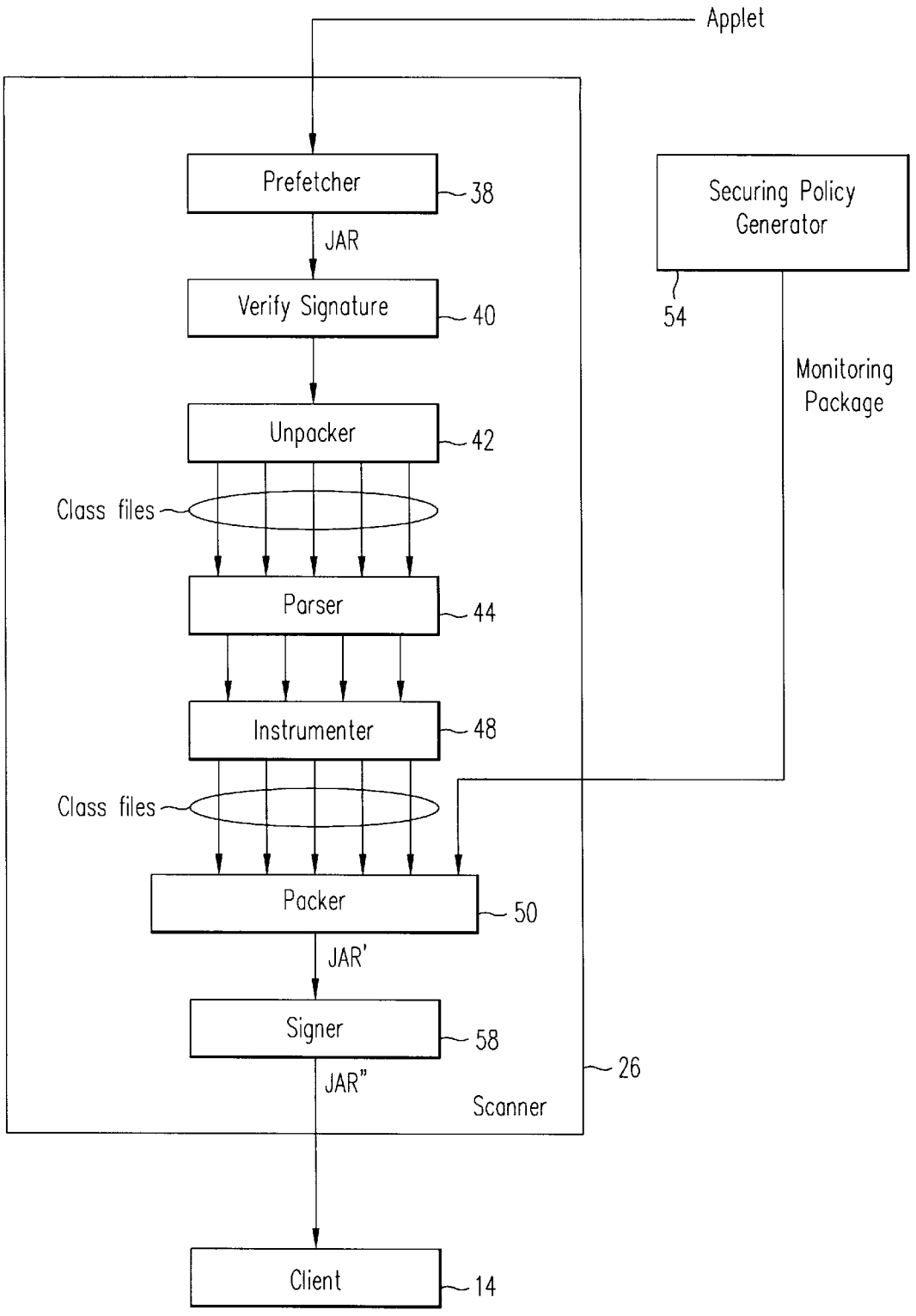


FIG. 2

COMPUTER NETWORK MALICIOUS CODE SCANNER

FIELD OF THE INVENTION

This invention pertains to computer networks and specifically to detecting and preventing operation of computer viruses and other types of malicious computer code.

BACKGROUND

With the rapid development of the Internet, Intranet, and network computing, applications (application programs) are distributed more and more via such networks, instead of via physical storage media. Many associated distribution technologies are available, such as Java and Active X. Therefore objects with both data and code flow around the network and have seamless integration with local computer resources. However, this also poses a great security risk to users. Code (software) from unknown origin is thereby executed on local computers and given access to local resources such as the hard disk drive in a user's computer. In a world wide web browser environment, such code is often automatically executed and the user might not even have a chance to be forewarned about any security risks (e.g. presence of computer viruses) he bears. Attempts have been made to reduce such risks; see Ji et al., U.S. Pat. No. 5,623,600, incorporated by reference in its entirety.

Active X technology, like Java, distributes code that can access local system resources directly. The web browser cannot monitor or block such accesses. Such an applet (application) can do virtually anything that a conventional program, for instance, a virus, is capable of doing. Microsoft Corp. and others have attempted to address this problem by using digital signature technology, whereby a special algorithm generates a digital profile of the applet. The profile is attached to the applet. When an applet is downloaded from the Internet, a verification algorithm is run on the applet and the digital profile to ensure that the applet code has not been modified after the signing. If an applet is signed by a known signature, it is considered safe.

However, no analysis of the code is done to check the behavior of the applet. It is not difficult to obtain a signature from a reputable source, since the signature can be applied for online. It has occurred that a person has created an Active X applet that was authenticated by Microsoft but contains malicious code. (Malicious code refers to viruses and other problematic software. A virus is a program intended to replicate and damage operation of a computer system without the user's knowledge or permission. In the Internet/Java environment, the replication aspect may not be present, hence the term "malicious code" broadly referring to such damaging software even if it does not replicate.)

Java being an interpreted language, Java code can be monitored at run-time. Most web browsers block attempts to access local resources by Java applets, which protects the local computer to a certain extent. However, as the popularity of Intranets (private Internets) increases, more and more applets need to have access to local computers. Such restrictions posed by the web browsers are becoming rather inconvenient. As a result, web browsers are relaxing their security policies. Netscape Communicator is a web browser that now gives users the ability to selectively run applets with known security risks. Again, decisions are made based on trust, with no code analysis done.

Hence scanning programs with the ability to analyze and monitor applets are in need to protect users.

At least three Java applet scanners are currently available commercially: SurfinShield and SurfinGate, both from Finjan, and Cage from Digitivity, Inc. SurfinShield is a client-side (user) solution. A copy of SurfinShield must be installed on every computer which is running a web browser. SurfinShield replaces some of the Java library functions included in the browser that may pose security risks with its own. This way, it can trap all such calls and block them if necessary.

SurfinShield provides run-time monitoring. It introduces almost no performance overhead on applet startup and execution. It is able to trap all security breach attempts, if a correct set of Java library functions is replaced. However, it is still difficult to keep track of the states of individual applets if a series of actions must be performed by the instances before they can be determined dangerous this way, because the scanner is activated rather passively by the applets.

Since every computer in an organization needs a copy of the SurfinShield software, it is expensive to deploy. Also, installing a new release of the product involves updating on every computer, imposing a significant administrative burden.

Because SurfinShield replaces library functions of browsers, it is also browser-dependent; a minor browser upgrade may prevent operation. SufinGate is a server solution that is installed on an HTTP proxy server. Therefore, one copy of the software can protect all the computers proxied by that server. Unlike SufinShield, SurfinGate only scans the applet code statically. If it detects that one or more insecure functions might be called during the execution of the applet, it blocks the applet. Its scanning algorithm is rather slow. To solve this problem, SurfinGate maintains an applet profile database. Each applet is given an ID which is its URL. Once an applet is scanned, an entry is added to the database with its applet ID and the insecure functions it might try to access. When this applet is downloaded again, the security profile is taken from the database to determine the behavior of the applet. No analysis is redone. This means that if a previously safe applet is modified and still has the same URL, SurfinGate will fail to rescan it and let it pass through. Also, because the size of the database is ever-growing, its maintenance becomes a problem over time.

Cage is also a server solution that is installed on an HTTP proxy server, and provides run-time monitoring and yet avoids client-side installations or changes. It is similar to X Windows. All workstations protected by the server serve as X terminals and only provide graphical presentation functionality. When an applet is downloaded to Cage, it stops at the Cage server and only a GUI (graphical user interface) agent in the form of an applet is passed back to the browser. The applet is then run on the Cage server. GUI requests are passed to the agent on the client, which draws the presentation for the user. Therefore, it appears to users that the applets are actually running locally.

This approach creates a heavy load on the server, since all the applets in the protected domain run on the server and all the potentially powerful computers are used as graphical terminals only. Also, reasonable requests to access local resources (as in Intranet applications) are almost impossible to honor because the server does not have direct access to resources on individual workstations.

These products fail to create any balance between static scanning and run-time monitoring. SurfinShield employs run-time monitoring, SurfinGate uses static scanning, and Cage utilizes emulated run-time monitoring. Since static scanning is usually done on the server and run-time monitoring on the client, this imbalance also causes an imbalance

between the load of the server and the client. To distribute the load between the client and the server evenly, the present inventor has determined that a combination of static scanning and run-time monitoring is needed.

SUMMARY

This disclosure is directed to an applet scanner that runs e.g. as an HTTP proxy server and does not require any client-side modification. The scanner combines static scanning and run-time monitoring and does not cause a heavy load on the server. It also does not introduce significant performance overhead during the execution of applets. The scanner provides configurable security policy functionality, and can be deployed as a client-side solution with appropriate modifications.

Thereby in accordance with the invention a scanner (for a virus or other malicious code) provides both static and dynamic scanning for application programs, e.g. Java applets or ActiveX controls. The applets or controls (hereinafter collectively referred to as applets) are conventionally received from e.g. the Internet or an Intranet at a conventional server. At this point the applets are statically scanned at the server by the scanner looking for particular instructions which may be problematic in a security context. The identified problematic instructions are then each instrumented, e.g. special code is inserted before and after each problematic instruction, where the special code calls respectively a prefilter and a post filter. Alternatively, the instrumentation involves replacing the problematic instruction with another instruction which calls a supplied function.

The instrumented applet is then downloaded from the server to the client (local computer), at which time the applet code is conventionally interpreted by the client web browser and it begins to be executed. As the applet code is executed, each instrumented instruction is monitored by the web browser using a monitor package which is part of the scanner and delivered to the client side. Upon execution, each instrumented instruction is subject to a security check. If the security policy (which has been pre-established) is violated, that particular instruction which violates the security policy is not executed, and instead a report is made and execution continues, if appropriate, with the next instruction.

More broadly, the present invention is directed to delivering what is referred to as a "live agent" (e.g., a security monitoring package) along with e.g. an applet that contains suspicious instructions during a network transfer (e.g. downloading to a client), the monitoring package being intended to prevent execution of the suspicious instructions. The suspicious instructions each may (or may not) be instrumented as described above; the instrumentation involves altering suspicious instructions such as by adding code (such as the pre-and post-filter calls) or altering the suspicious instructions by replacing any suspicious instructions with other instructions.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** shows diagramatically use of a scanner in accordance with this invention.

FIG. **2** shows detail of the FIG. **1** scanner.

DETAILED DESCRIPTION

Several characteristics of the well known Java language and applets are pertinent to the present scanning method and apparatus. Java is an interpreted, dynamic-linking language. Only the application modules are distributed, and all the standard library functions are provided by the interpreter, for instance a web browser. Because Java byte code is platform-independent, applets have to use some of the standard library functions to access operating system resources.

This creates two opportunities in accordance with the invention to detect attempts to use operating system resources. First, one can "trick" applets into calling particular functions supplied by the scanner during the dynamic linking stage. This is done by replacing the browser Java library routines with the scanner's monitoring routines of the same name. Second, since invocations of such functions have to be resolved at run-time, symbolic names of these functions are kept in the Java applet module. The scanner can detect possible use of these functions by looking at the static code itself. The first opportunity provides run-time monitoring. It is the most definitive method to determine the security risks posed by an applet.

The second opportunity enables statically scanning an applet, without running it, to detect possible security risks. If a set of insecure functions is properly defined and an applet never calls any function in the set, the applet can be assumed to be safe. However, this static scanning method is not definitive, since an applet might show different behavior given different user input. Under certain conditions, the instruction in the applet that makes the function call may never be executed. If static scanning is used without run-time monitoring, many such "false alarms" of security risks are produced undesirably.

After the code of an applet is downloaded, e.g. via the Internet to a client platform (local computer), an instance of the applet is created in the conventional Java "virtual machine" in the web browser (client) running on that local computer. Different instances of the same applet might produce different results given different inputs. A running instance of an applet is conventionally called a session; sessions are strictly run-time entities. Static scanning cannot analyze sessions because static scanning does not let the applet run. Sessions are important because an instance of an applet will often perform a series of suspicious tasks before it can be determined dangerous (i.e., in violation of the security policy). Such state information needs to be associated with the sessions. The present applet scanner thereby stops sessions instead of blocking execution of the entire applet.

A security policy defines what functions an applet needs to perform to be considered a security risk. Examples of security policies include preventing(1) applets from any file access, or (2) file access in a certain directory, or (3) creating certain Java objects. An applet scanner in accordance with the invention may allow different security policies for different clients, for different users, and for applets from different origins.

FIG. **1** is a high level block diagram illustrating the present scanner in the context of conventional elements. The Internet (or an Intranet) is shown generally at **10**. The client machine or platform (computer) **14**, which is typically a personal computer, is connected to the Internet **10** via a conventional proxy server machine (computer) **20**. Client machine **14** also includes local resources **30**, e.g. files stored on a disk drive. A conventional web browser **22** is software that is installed on the client machine **14**. It is to be understood that each of these elements is complex, but except for the presently disclosed features is conventional.

Upon receipt of a particular Java applet, the HTTP proxy server **32**, which is software running on server machine **20**

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.