US007284274B1

(12) **United States Patent**
Walls et al.

(10) **Patent No.: US 7,284,274 B1**
(45) **Date of Patent: Oct. 16, 2007**

(54) **SYSTEM AND METHOD FOR IDENTIFYING AND ELIMINATING VULNERABILITIES IN COMPUTER SOFTWARE APPLICATIONS**

(75) Inventors: **Thomas J. Walls**, East Setauket, NY (US); **Viren Shah**, Ashburn, VA (US); **Anup K. Ghosh**, Centreville, VA (US)

(73) Assignee: **Cigital, Inc.**, Dulles, VA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 682 days.

(21) Appl. No.: **10/050,764**

(22) Filed: **Jan. 18, 2002**

Related U.S. Application Data

(60) Provisional application No. 60/262,085, filed on Jan. 18, 2001.

(51) **Int. Cl.**
G06F 15/18 (2006.01)

(52) **U.S. Cl.** **726/25**; 726/22

(58) **Field of Classification Search** 713/201; 726/25

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,924,408	A	*	5/1990	Highland	706/60
4,941,102	A	*	7/1990	Darnell et al.	706/60
5,132,972	A	*	7/1992	Hansen	714/38
5,133,045	A	*	7/1992	Gaither et al.	706/46
5,247,693	A	*	9/1993	Bristol	717/139
5,293,629	A	*	3/1994	Conley et al.	717/154
5,500,941	A	*	3/1996	Gil	714/38
5,581,696	A	*	12/1996	Kolawa et al.	714/38
5,699,507	A	*	12/1997	Goodnow et al.	714/38
5,761,408	A	*	6/1998	Kolawa et al.	714/38
5,784,553	A	*	7/1998	Kolawa et al.	714/38
5,842,019	A	*	11/1998	Kolawa et al.	717/130
5,850,516	A	*	12/1998	Schneier	726/25
5,854,924	A	*	12/1998	Rickel et al.	717/132
5,860,011	A	*	1/1999	Kolawa et al.	717/142
5,922,079	A	*	7/1999	Booth et al.	714/26
5,925,123	A	*	7/1999	Tremblay et al.	712/212
5,970,242	A	*	10/1999	O'Connor et al.	717/100
6,014,723	A	*	1/2000	Tremblay et al.	711/1
6,085,029	A	*	7/2000	Kolawa et al.	714/38
6,125,439	A	*	9/2000	Tremblay et al.	712/202
6,148,401	A	*	11/2000	Devanbu et al.	713/170
6,154,876	A	*	11/2000	Haley et al.	717/133
6,381,698	B1	*	4/2002	Devanbu et al.	713/170

(Continued)

OTHER PUBLICATIONS

D. Wagner, J. Foster, E. Brewer, and A. Aiken. A first step towards automated detection of buffer overrun vulnerabilities. In Network and Distributed System Security Symposium, San Diego, CA, Feb. 2000.*

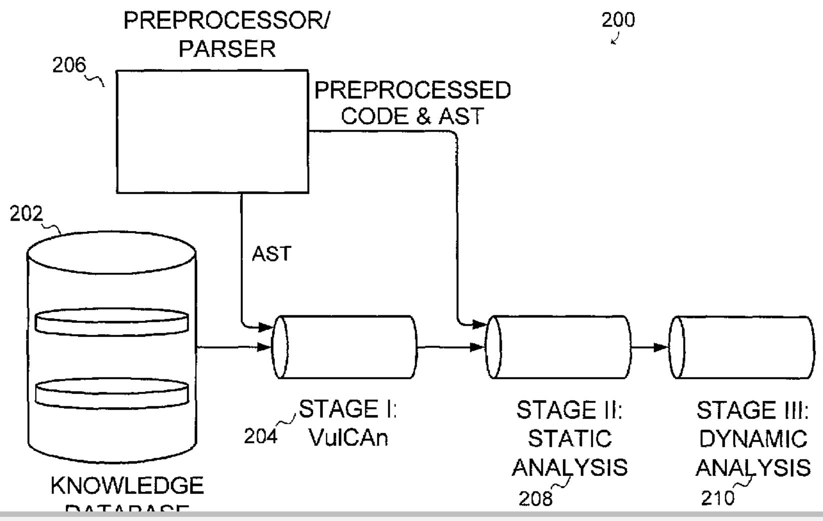
(Continued)

Primary Examiner—Nasser Moazzami
Assistant Examiner—David Garcia Cervetti
(74) *Attorney, Agent, or Firm*—Edell, Shapiro & Finnan, LLC

(57) **ABSTRACT**

A system and method for certifying software for essential and security-critical systems. The system and method provide a methodology and corresponding analysis engines increase the level of confidence that common vulnerabilities are not present in a particular application. A pipeline system consisting of independent modules which involve increasingly complex analysis is disclosed. The pipeline approach allows the user to reduce computation time by focusing resources on only those code segments which were not eliminated previously in the pipeline.

11 Claims, 2 Drawing Sheets



U.S. PATENT DOCUMENTS

6,412,071	B1 *	6/2002	Hollander et al.	726/23
6,427,232	B1 *	7/2002	Ku et al.	717/124
6,473,896	B1 *	10/2002	Hicken et al.	717/132
6,513,154	B1 *	1/2003	Porterfield	717/101
6,578,094	B1 *	6/2003	Moudgill	710/57
6,584,569	B2 *	6/2003	Reshef et al.	726/25
6,718,485	B1 *	4/2004	Reiser	714/38
6,806,893	B1 *	10/2004	Kolawa et al.	715/848
6,807,569	B1 *	10/2004	Bhimani et al.	709/217
6,862,696	B1 *	3/2005	Voas et al.	714/38
6,895,578	B1 *	5/2005	Kolawa et al.	717/130
2001/0013094	A1 *	8/2001	Etoh et al.	712/227
2002/0026591	A1 *	2/2002	Hartley et al.	713/201
2003/0233581	A1 *	12/2003	Reshef et al.	713/201
2004/0006704	A1 *	1/2004	Dahlstrom et al.	713/200
2004/0103315	A1 *	5/2004	Cooper et al.	713/201

OTHER PUBLICATIONS

Necula et al., The design and implementation of a certifying compiler, 1998, ACM, pp. 333-344.*

Bush et al., A static analyzer for finding dynamic programming errors, 2000, Software Practice and Experience, pp. 775-802.*

Viega, J.; Bloch, J.T.; Kohno, Y.; McGraw, G., ITS4: a static vulnerability scanner for C and C++ code, Dec. 2000, IEEE, pp. 257-267.*

D. Wagner, J. Foster, E. Brewer, and A. Aiken. A first step towards automated detection of buffer overrun vulnerabilities. In Network and Distributed System Security Symposium, San Diego, CA, Feb. 2000.*

Necula et al., The design and implementation of a certifying compiler, 1998, ACM, pp. 333-344.*

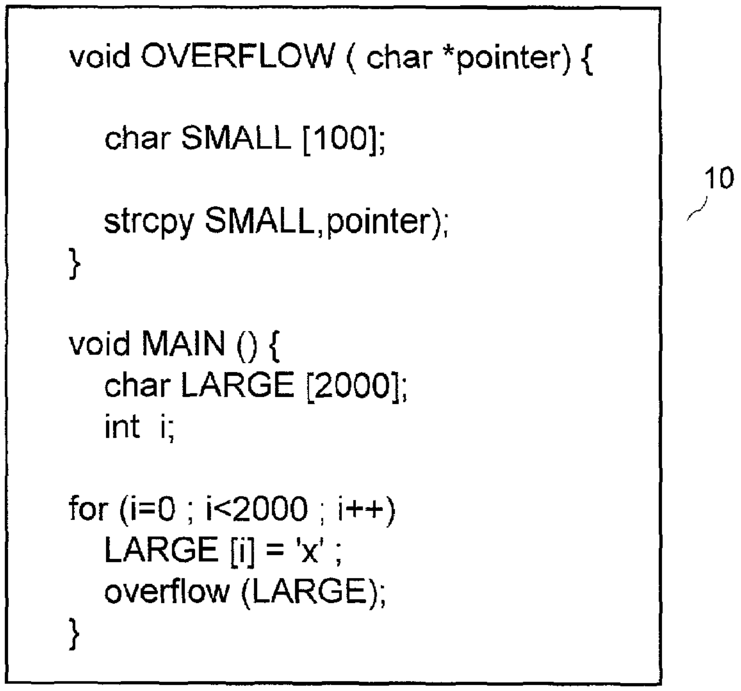
Bush et al., A static analyzer for finding dynamic programming errors, 2000, Software Practice and Experience, pp. 775-802.*

Viega, J.; Bloch, J.T.; Kohno, Y.; McGraw, G., ITS4: a static vulnerability scanner for C and C++ code, Dec. 2000, IEEE, pp. 257-267.*

Ghosh et al., An Approach for Certifying Security in Software Components, 1998, Proc. 21st {NIST}-{NCSC} National Information Systems Security Conference, <citeseer.ist.psu.edu/104720.html>.*

Ghosh, A.K.; O'Connor, T.; McGraw, G., An automated approach for identifying potential vulnerabilities in software, 1998, Security and Privacy, 1998. Proceedings. 1998 IEEE Symposium on, May 3-6, 1998 pp. 104-114.*

* cited by examiner



```
void OVERFLOW ( char *pointer) {

   char SMALL [100];

   strcpy SMALL,pointer);
}

void MAIN () {
   char LARGE [2000];
   int  i;

for (i=0 ; i<2000 ; i++)
   LARGE [i] = 'x' ;
   overflow (LARGE);
}
```

FIG. 1A

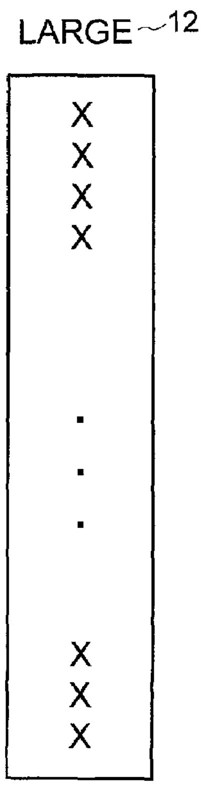


FIG. 1B

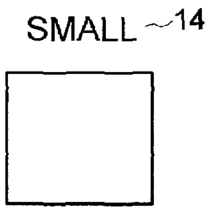


FIG. 1C

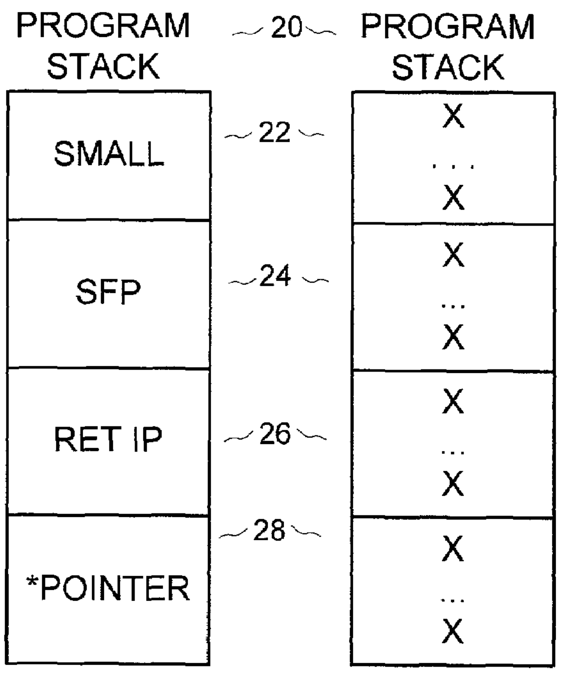


FIG. 1D

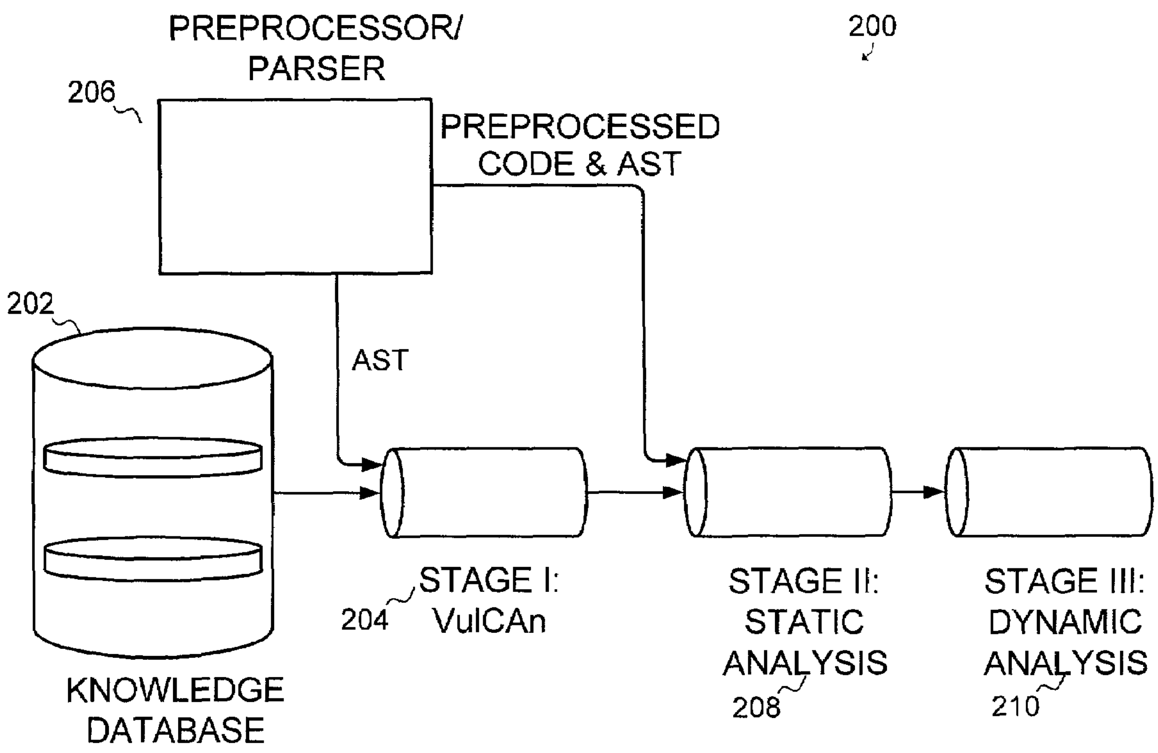
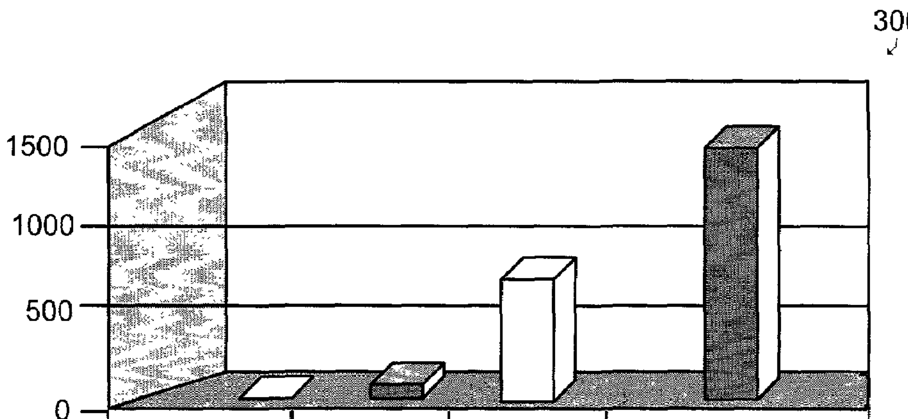


FIG. 2



	GREP	STAGE I VULCAN	STAGE II STATIC ANALYSIS	STAGE III DYNAMIC ANALYSIS (PROJECTED)
▨ MINUTES	3	4	900	1500

FIG. 3

SYSTEM AND METHOD FOR IDENTIFYING AND ELIMINATING VULNERABILITIES IN COMPUTER SOFTWARE APPLICATIONS

This application claims the benefit of U.S. Provisional Application No. 60/262,085, filed Jan. 18, 2001, which is herein incorporated by reference in its entirety.

This invention was made with Government support under Cooperative Agreement No. 70NANB7H3049 administered by the National Institute for Standards and Technology. The Govemment has certain rights in the invention.

BACKGROUND

1. Field of the Invention

The present invention relates generally to computer system security, integrity, and reliability. More particularly, the present invention relates to examination and certification of software application programs to identify and eliminate vulnerabilities arising from poor software application programming techniques.

2. Background of the Invention

The explosion of electronic commerce has placed computer software applications at the cornerstone position of on-line business. Software is the brick and mortar of the new economy, but the migration from physical to virtual retail space has placed both the consumer and vendor at risk in unforeseen ways. If the new economy is going to survive, software will have to become more resistant to attack and will have to continuously improve to meet the rigorous demands of an on-line market.

An example of the magnitude of the problems faced by software users is illustrated by the distributed denial-of-service (dDoS) attacks against major e-commerce sites of February, 2000. Some of the brightest luminaries in e-commerce, including Yahoo!, Amazon.com, Buy.com, ZDNet, and CNN.com were effectively taken down for a period of hours by these attacks. What is most impressive and disturbing about these attacks is that they were against very high volume sites. For instance, according to Media Metrix, an online traffic measurement firm, Yahoo! had more unique visitors in January 2000 than any other online site. The other victims were among the top fifty sites. The dDoS attacks were able to bombard these sites with data at rates of up to one gigabit of data per second. The collective downtime of these sites resulted in a loss of revenue estimated to be in the millions of U.S. dollars.

Though denial-of-service (DoS) attacks often exploit weaknesses in protocols to hold network services hostage to the attacks, what is often overlooked by analysts is that such dDoS attacks are often made possible by flaws in software. A key to implementing an effective dDoS attack is to compromise a very large number of machines in order to plant the dDoS attack software, which, in the February attacks, went by names such as Trinoo or TFN2000. Sites are usually compromised in the first place by exploiting some flaw in software. In the case of many dDoS attacks, Unix servers are compromised, often by a well-known flaw in the Remote Procedure Call (RPC) service. Once a site is compromised and the malicious software installed, the compromised site becomes a zombie that acts maliciously on command at some future time. One of the keys to preventing these types of attacks in the future is to secure the software on the server systems such that they are not vulnerable to compromise in the first place.

under any conditions, including normal operation, as well as unusual or attack conditions, can result in immediate loss of revenue, as well as jeopardizing the long-term viability of the business. For instance, well-known flaws in CGI scripts have enabled hackers to alter Web pages with political messages. If the Web pages of a financial investment firm were vandalized, investors and Wall Street would likely lose confidence in the ability of the firm to securely manage the assets of firm's investors.

For companies that develop and release application software, the expense in adequately addressing security vulnerabilities is very high. Moreover, for any vulnerabilities that were not adequately foreseen, there will be a corresponding drop in consumer confidence which cannot be measured. For example, both Netscape and Microsoft experienced well-publicized security-related flaws in their Internet browsers in 1997.

Developers of operating systems such as Sun Microsystems and Hewlett-Packard also spend considerable human resources tracking bugs that have the potential to be security flaws in their commercial operating systems. Such costs are often transferred to the end users, either directly (in the form of increased software prices) and indirectly (in the form of increased maintenance costs). It is well-known that the time and expense involved for system administrators to patch, upgrade, and maintain the security of computer systems is very high and increases with both new software additions and more sophisticated attacks.

The buffer overrun attack is one of the most pervasive modes of attack against computer systems today. Probably the most infamous buffer overrun attack is the Morris worm of 1988 that resulted in the shutdown of a significant portion of the Internet infrastructure at the time (which consisted of primarily university and government nodes). The worm was a self-propagating buffer overrun attack that exploited a program vulnerability in the Unix fingerd network service. The worm illustrated the serious nature of software flaws and how they can be leveraged to breach security on other systems.

Since the Morris worm, buffer overrun attacks have become a very popular method of breaking into systems or obtaining super user privilege from user-level accounts. According to statistics released by the Computer Emergency Response Team (CERT) Coordination Center of Carnegie Mellon University's Software Engineering Institute, about 50 percent of computer incidents reported today in the field involve some form of buffer overrun.

To further complicate the problems presented with application software, unsafe languages, such as C, make buffer overflow attacks possible by including standard functions, such as, for example, gets, strcat, and strcpy, that do not check the length of the buffer into which input is being copied. If the length of the input is greater than the length of the buffer into which it is being copied, then a buffer overflow can result. Safe programming practices that allow only constrained input can prevent a vast majority of buffer overflow attacks. However, many security-critical programs already in the field today do not employ safe programming practices. In addition, many of these programs are still coded in commercial software development labs in unsafe languages today.

As described above, buffer overrun attacks are made possible by program code that does not properly check the size of input data. When input is read into a buffer and the length of the input is not limited to the length of the buffer

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.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.