REDESIGN OF A DISTRIBUTED VULNERABILITY SCANNING TOOL TO USE AN ABSTRACT, VENDOR-INDEPENDENT PKI INTERFACE

M.R. Crocker
m.crocker@computer.org

G.S. Knight
Royal Military College of Canada
knight-s@rmc.ca

Abstract

Detecting security vulnerabilities in large heterogeneous networks is generally accomplished using either host-based or network-based security auditing tools. A host-based security auditing tool runs on the host being assessed and has the advantage of seeing the vulnerabilities of a machine from the inside. A network-based tool audits hosts on network by sending crafted packets; it sees the machines as a remote attacker would see them. Sun Microsystems, Inc. is developing a distributed security vulnerability scanner, Bruce, that seeks to combine the strengths of the host-based perspective with the scalability and centralized information collection of a network-based tool. Host-based vulnerability assessment tests can be distributed to network hosts, and vulnerability reports are collected centrally by a security analyst. Bruce is in beta version and the source code has been made publicly available under Sun Community Source Licensing.

This work addresses the confidentiality and integrity of the vulnerability reports in the Bruce tool. There is a perceived weakness in how these reports are requested, stored, and communicated to the security analyst's workstation. A prototype solution has been developed and implemented to mitigate this weakness. The solution modifies the Bruce tool by introducing in its design a standard modular interface for security services. This revised security services architecture provides enhanced protection of network vulnerability information by using strong standardized PKI mechanisms via a vendor independent interface. A version of the required security services module has been implemented using the Entrust® Public Key Infrastructure Toolkit and API.

Technical level of this paper is - 3

REDESIGN OF A DISTRIBUTED VULNERABILITY SCANNING TOOL TO USE AN ABSTRACT, VENDOR-INDEPENDENT PKI INTERFACE

1. INTRODUCTION

Society has come to rely more heavily on computer applications and networks, and the risk of significant damage resulting from a correctable vulnerability is significant. Although this situation is being mitigated by security education and increased adoption of both preventive and reactive security systems, such as vulnerability scanners and intrusion detection systems, the situation is still very serious [JONES]. The vulnerabilities existing on a machine and within a network must be determined quickly and accounted for by repair or at the very least auditing and contingency planning. In protecting networks and individual hosts against malicious intents, vulnerability scanners are becoming critical tools for network administrators. Once an administrator becomes aware of a vulnerability, it is important to determine quickly and securely if the conditions are present in the network and on specific hosts that would allow the vulnerability to be exploited. In a large heterogeneous network environment, this is difficult.

A unique distributed vulnerability scanner is being developed by Sun Microsystems Inc. that seeks to combine the design strengths of both host-based and network-based vulnerability scanning tools. The source for the vulnerability scanning tool was available but no higher level design documentation. Design documentation was reverse engineered from the source code. The source code and recovered design were inspected. The results of the inspection identified perceived security and design weaknesses that could impact the tool's suitability as an enterprise vulnerability scanner in its current state. Also, there are many components of the current tool that implement and use security services. The provision of these services should be separated from the vulnerability scanner business logic and consolidated in a coherent module with a well-defined interface.

The complexity of handling security problems such as integrity, confidentiality and authentication in software can be simplified using the security services provided by a Public Key Infrastructure (PKI). Several security based software companies including Entrust® and Verisign® have created substantial PKIs that provide well conceived and implemented security services that are accessible through application programming interfaces (APIs). Developers can deal with security problems in their software applications by integrating their applications with the security services provided by a PKI. This allows a developer to make use of the security expertise used to develop the solutions available through a PKI and to focus more energy on the business logic of the application.

The Sun vulnerability scanner was redesigned to provide security services via a consolidated vendor independent interface. A package using the Entrust API was implemented to provide the required services.

Section 2 of the paper is a review of vulnerability scanning technologies. Section 3 introduces the Sun vulnerability scanning tool and the redesign of the tool is presented in section 4. Section 5 is a brief conclusion to the paper.

2. VULNERABILITY SCANNERS

Humphries et al broke down vulnerabilities into five different categories: Physical/Environmental; Network/Connectivity; Platform/Operating System; Application/Service; and Human Policy [HUMPH]. When considering security-auditing tools, it is helpful to look at vulnerabilities in the taxonomy described by Humphries et al. It provides a straightforward guide to understanding what sort of checks need to be performed when verifying the security posture of computer systems and networks. Automated tools deal mostly with scanning for vulnerabilities associated with network connections, operating systems and applications/services. It is recognized that physical and policy vulnerabilities could have significant affect on a system's overall security, however they do not lend themselves to easy assessment using automation techniques.

The majority of available tools fall into the broad categories of host-based or network-based vulnerability scanners. These two types of tools have a number of fundamental differences and lend themselves to different deployments in an organization.

Host-based Vulnerability Scanners

A host-based vulnerability scanner (HVS) runs on a specific host and is able to test the host for vulnerabilities unique to the operating system, resident applications, local services available, hardware configuration and local user profiles and activities. It generally consumes a small amount of disk space and varies in resource consumption depending on the configuration of the tool. The checks can be matched to the exact circumstances of the machine and, if configured properly, can produce very useful results. There are a number of HVS's available including Titan [TITAN], the Computer Oracle and Password System [COPS], Tiger [TIGER] and a few commercial products available from the major computer security vendors.

The strength of an HVS lies in the tool's inside perspective of the system. These tools are generally run with full system access and consequently can examine any file or resource on the machine being tested. An HVS can identify risky user activity, to some extent it can act as an intrusion detection system to discover compromise, and it has the ability to examine security relevant systems/services in low-level detail to uncover vulnerabilities.

HVS's can be tailored very precisely to the individual user configurations and specific operating system of the target machine. If a system administrator feels a specific user is a risk to the security of the system, he can adjust the configuration of the HVS to look for specific vulnerabilities consistent with the risk the user poses. An HVS can automatically verify the strength of user passwords by scanning password files and running the results through programs such as Crack [CRACK]. An HVS can verify an operating system's configuration. Such checks might include for example: checking for shared disk drives that a user may have made publicly accessible without being aware of the possible repercussions.

An HVS can scan system logs looking for abnormal user activity, or perhaps gaps in activity, that might indicate a vulnerability that had been exploited. It can look for suspicious file names, known backdoor programs or abnormal file attributes. It can compare the current state of files with a known safe baseline, similar to the functionality offered by Tripwire [TRIP]. Although not intended to perform intrusion detection in most cases, the presence of a backdoor program could indicate that a vulnerability had been exploited and might elicit more careful monitoring until the nature of the compromise is discovered and corrected.

An HVS can be configured to go into significant detail in evaluating the host system because of its full system access. If you consider for example that an attacker might be able to exploit an old dormant account for malicious purposes. A network-based system would generally be incapable of determining that this vulnerability existed while an HVS could reasonably easily determine that this vulnerability is present and further, that such an account had been recently used. An HVS tool can be configured to check for installed devices that a user might have inadvertently or maliciously activated such as a modem on an active phone line. After a complete run with proper configuration, the administrator can be provided with some level of assurance that the system is not vulnerable to a known set of exploits. The key to success with an HVS is the development of a library of vulnerability checks for well known and recently discovered vulnerabilities.

The predominant weaknesses of an HVS are the tool's scalability and lack of network visibility. Scanning many machines might require different configurations for different machines and changing the configurations in the course of normal installation and maintenance of the vulnerability scanners is time-consuming. Beyond the difficulties of configuration, collation of the vulnerabilities of all the hosts on a large network is very difficult. The results produced by the scanners have to be analyzed with respect to the specifics of the machine and the configuration of the tool on that host. A mapping of the results to the scanning tool configuration at the time would be necessary for useful results. An HVS is generally only configured to examine the files and services existing on the host and will not initiate network communication in order to gain network visibility [TITAN][COPS][TIGER]. In general, an HVS scales very poorly in a large heterogeneous network and because of its lack of network visibility and result collation problems, it would be difficult to gain a system-wide understanding of vulnerabilities.

Network-based Vulnerability Scanners

A network-based vulnerability assessment scanner (NVS) is run from a central machine or suite of machines against remote hosts across a network. The software sees the target machine and network in the same way as a network-based hacker would see the machine, and assesses the target for a number of known vulnerabilities. There is no executable code placed on the target but rather a series of specially crafted packets are sent to the target. Depending on the response or lack of response from the target, vulnerabilities are deduced. Each time a new vulnerability is published by the security community, the developers of an NVS attempt to create a method to check for the vulnerability. Collection of vulnerability information across the network is quick and, if properly configured, the tool can be very effective. There are a number of NVS's available including the Security Administrator Tool for Analyzing Networks [SATAN], the Security Auditor's Research Assistant [SARA], and Nessus [NESSUS].

NVS's generally have two main strengths: central coordination; and platform independence. The software can be configured and run from a central location and the results are collated quickly at a central location. This allows for large-scale scanning and, when considering assessing vulnerabilities for a large network, scalability is critical. In addition to simply providing vulnerability information for the known machines on a network, most of the tools can be configured to check for other devices existing on the network being scanned. An NVS can check for vulnerabilities that pertain to routers, networked printers, switches and new hosts that have been recently added to the network [ISS]. This allows an administrator to gain an accurate picture of the current topology of the network and the collective vulnerabilities of its constituents. Although the software itself is usually written to be loaded on a specific platform, the targets that are being assessed for vulnerabilities can generally be using one of a number of different operating systems. Many of the tools run hundreds of checks against target machines and it is reasonably easy to configure the software to scan a large heterogeneous network.

In addition to their strengths, NVS's also have a number of weaknesses that result from the overall design philosophy of the type of tool. NVS's work at the network interface and cannot check for vulnerabilities that might be accessible to a user who has physical access to the machine or a compromised remote account. If a malicious hacker has manipulated a system to introduce a 'backdoor' for remote login, an NVS may not be capable of determining this. The tool does not check for machines with both network card and modem access as a potential source of problems. A network that was being illegitimately accessed through the modem of a single host may go unnoticed during an NVS scan. In general, an NVS can only assess a host for a smaller number of vulnerabilities that can be assessed by an HVS.

Distributed Vulnerability Scanner

Distributed vulnerability scanners seek to combine the strengths of the host-based perspective with the scalability and centralized information collection of a network-based tool. Sun Microsystems Inc. is developing a tool as part of their Sun Enterprise ™ Network Security Service (SENSS) called Bruce. It is described as a "networked host-vulnerability scanner." The source code has been made publicly available for scrutiny under the terms of the Sun Community Source License and the application appears to offer functionality not available in commercial off-the-shelf (COTS) software.

3. BRUCE

Description

Bruce is a complex networked host vulnerability scanner that uses a Java-based distributed architecture to distribute and run executable vulnerability checks on a network of hosts. After installation Bruce does not require user intervention in the network to function. The Bruce application uses a hierarchical communication topology for distributing the vulnerability checks and then collecting the results. A vulnerability detection agent is dispatched from a parent to a child host and is then duplicated at the child host for further movement down a hierarchical tree, each child becoming a parent to its children. The results are collected in a reverse process of parents polling their children for the vulnerability data in a recursive fashion until such time as the data has been pulled to the top of the hierarchy (see Figure 1). Note that this is a logical communication hierarchy and in no way constrains the underlying physical topology of the network.

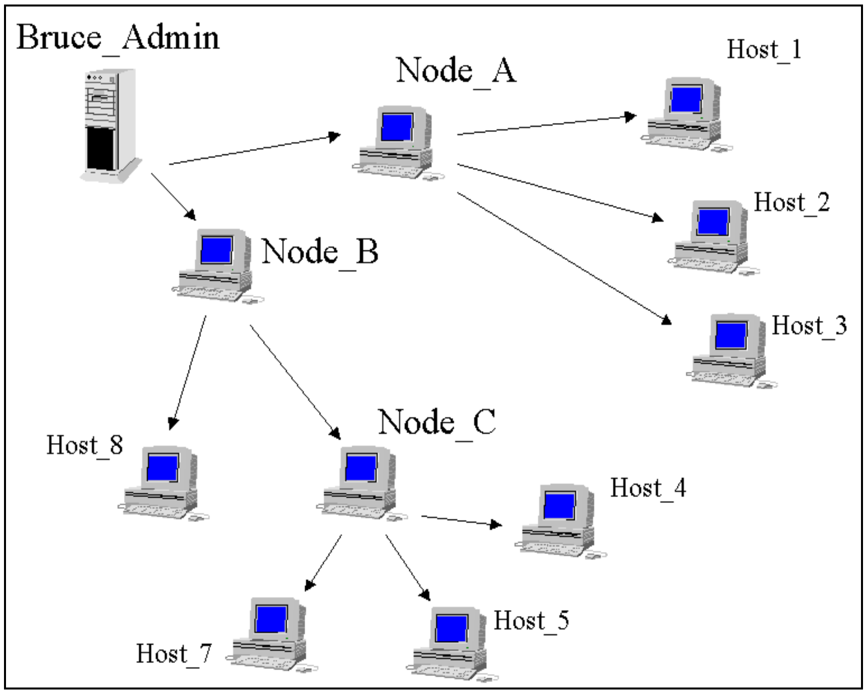


Figure 1 - Bruce Hierarchy

A very simple description of Bruce functionality is required in order to describe the perceived security weaknesses and to understand the proposed solution. Digitally signed vulnerability checks composed of various data configuration elements and executable content are passed from the Bruce_Admin machine to his logical children, in this case Node_A and Node_B. All communication is initiated from the parent to the child. Bruce uses Java Remote Method Invocation (RMI) for communications but applies the additional security of a pluggable secure transport module between RMI (application layer) and the transport layer.

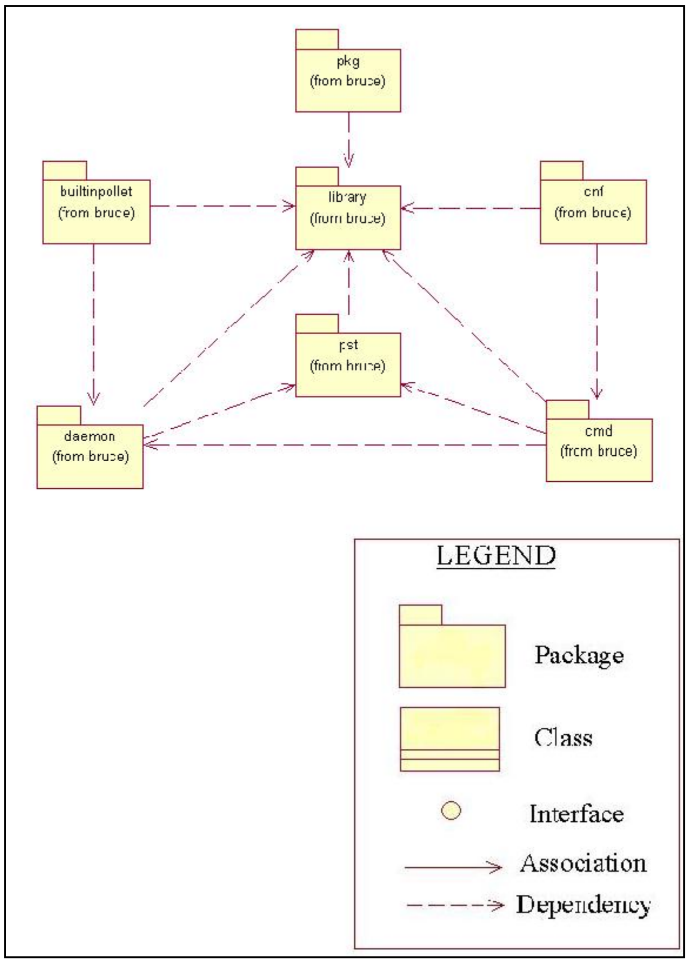
As the vulnerability check is launched, daemon processes running on both Node_A and Node_B are activated and receive the executable content and configuration data contained in the vulnerability check. The daemon process on Node_A in turn passes the vulnerability check on to its three hosts: Host_1, Host_2 and Host_3. The daemon process on Node_B passes the check on to Node_C and Host_8. Node_C in turn passes the check on to its children. Once the vulnerability check has been successfully passed from parent to child, communications are terminated on that link and the executable vulnerability check is performed on the hosts and nodes. It is possible to exempt a host from execution of a particular check.

Each of the hosts can have different operating systems but must have a Java Virtual Machine in order to support the Bruce architecture. Based on the configuration data contained in the vulnerability check, hosts will run only the executable content appropriate for their environment. After execution, the results are stored on the local host in clear format in a designated file and are later claimed in a polling process very similar to that used to distribute the vulnerability check. Based on timings, each parent will poll its children and if the child has completed the check, will retrieve the results. The results will eventually be deleted based on a configuration setting and each parent has the vulnerability results for its children, and recursively their children, stored locally until deletion. Eventually the vulnerability results will bubble to the top of the Bruce logical hierarchy.

Design

Bruce is an object-oriented application based largely on Java 2 technology. The source code unpacks into seven different packages: builtinpollet, cmd, cnf, daemon, library, pkg, and pst. In order to understand the architecture an implementation of the tool, the source code was reverse engineered using Rational Rose® to provide UML design documentation. A multi-layer logical model has been specified that consists of a hierarchical set of class diagrams and class specifications. Collaboration diagrams have been constructed to describe the major scenarios of the scanner business logic. The recovered design documentation and source code were inspected. Figure 2 below is a graphical depiction of the high level packages. The dependencies drawn between packages indicate that classes of one package either depend or have an association with the classes of another package.

Perceived Weakness



In its original design, Bruce implements a number of services that allow it to cryptographically sign objects, produce certificates, and manage cryptographic keys. These are services that require very careful and knowledgeable implementation. It might be better to leverage the development investment of a commercial PKI vendor to provide these services and to separate the cryptographic services from the vulnerability scanning business logic. However, the source code performs a security related functions in various places and there is no single package that could be adjusted to link the application to the services provided by a PKI. There should really be a separation of concerns in the design of the scanner.

Another significant weakness is the confidentiality of the vulnerability reports. This involves the vulnerability reports being passed in clear text format between child and parent throughout the network. Based on limitations imposed by US cryptography export regulations in effect at the time of development, the developers of Bruce elected to delay solving the problem of output reports being stored locally and passed over the network in plaintext format. An attacker can sniff the network outside the target machine, extracting all packets from the communication channel and reassembling them into a readable format. This vulnerability information is valuable to a potential attacker.

The third significant weakness is the integrity of the vulnerability reports. The application is currently vulnerable to a man-in-the-middle attack. If an attacker could sit between a communicating parent and child, it could learn all of the vulnerabilities of the child. In fact, an attacker could conceivably adjust the pollet output reports as they transition and provide false information to the parent. In Bruce v1.0 EA4, the pollet output reports are not digitally signed and there is no method that ensures the integrity of the report from the point of generation. Similarly, in the current output collection scheme, if the host is compromised it will also have access to the vulnerability information for all its children. This is a dangerous situation. The Bruce administrator can harden the machines within his control, but he cannot necessarily dictate the security on machines that are levels below him in the Bruce hierarchy. The Bruce application could actually help an attacker, who had compromised a secondary server, to gain vulnerability information on many machines very easily. An attacker could alter the pollet output reports and misrepresent the actual vulnerabilities to the next level up in the Bruce hierarchy. Not only would the attacker have gained knowledge about the vulnerabilities of machines below him on the network, but the Bruce administrators would be unaware of the problem and would assume, based on the reports, that all was well.

Fortunately, a PKI offers solutions to problems of confidentiality and integrity. Instead of developing a unique solution for the Bruce application to deal with these security problems it is proposed to address the deficiencies using a standardized set of PKI services.

4. REDESIGN OF BRUCE

Security Services Interface

The quality characteristics of security, functionality, reliability, usability, efficiency, maintainability, portability, ease of administration and flexibility to change were considered in developing a solution to the problems of confidentiality and integrity. The prototype developed here uses the Entrust PKI but a different PKI could be plugged in with minor changes.

The prototype design is based on the concept of using the Façade pattern as an interface to a PKI. The Façade pattern is described by Gamma et al [GAMMA] and further explored in the Java language by Eckel [ECKEL_PAT]. The Façade pattern provides a unified interface to a set of interfaces in a sub-system. This interface removes some of the complexity in dealing with the sub-system and also allows easy substitution of different sub-system implementations (in this case, a different PKI). Figure 3 below shows the high-level design of the prototype regarding the Bruce application's interaction with the PKI. The diagram indicates the directional associations (solid lines) and dependencies (broken lines) between classes (boxes). The association linking the PKIFacade class back to itself is a result of use of the singleton pattern [ECKEL_PAT] that limits creation of these types of objects to a single instance.

In Figure 3, the *PKIFacade* class in the center of the diagram acts as the façade to provide an interface with the classes of the *entrustpki* package in order to access the security services of the PKI. Changing to a different PKI would require the creation of a different package to replace the *entrustpki* package and some minor modifications to the *pkifacade* class, no changes would be required in the modified Bruce source code.

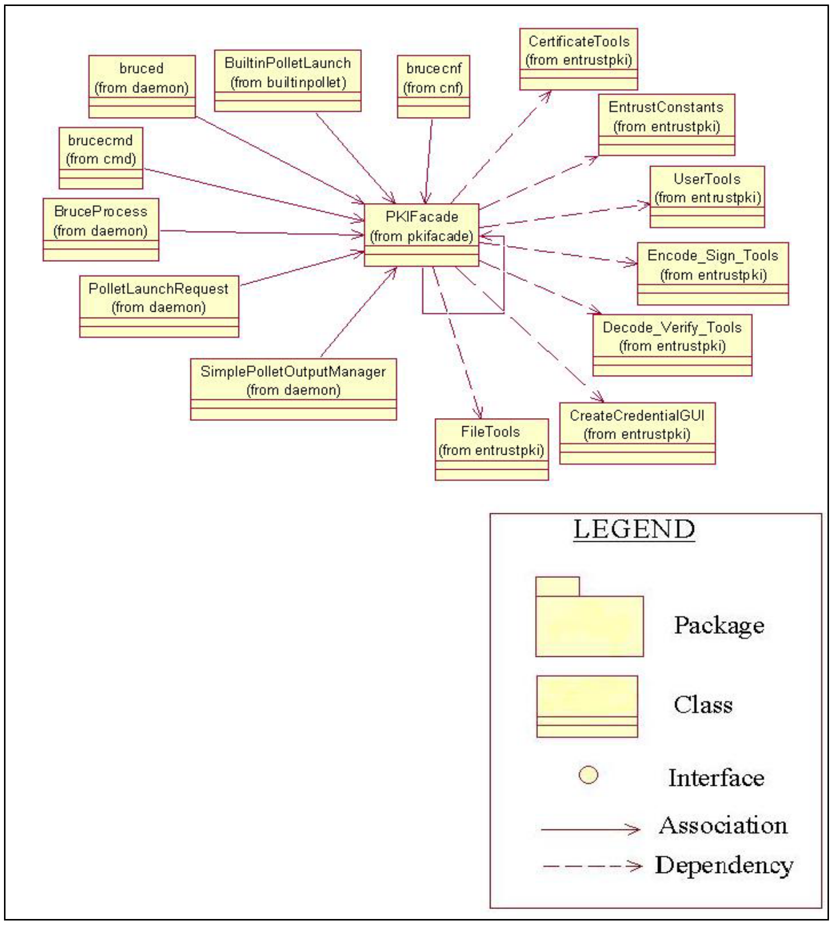

bruced (from daemon)
BuiltinPolletLaunch (from builtinpollet)
brucecnf (from cnf)
CertificateTools (from entrustpki)
EntrustConstants (from entrustpki)
brucecmd (from cmd)
UserTools (from entrustpki)
BruceProcess (from daemon)
PKIFacade (from pkifacade)
Encode_Sign_Tools (from entrustpki)
PolletLaunchRequest (from daemon)
Decode_Verify_Tools (from entrustpki)
SimplePolletOutputManager (from daemon)
CreateCredentialGUI (from entrustpki)
FileTools (from entrustpki)
LEGEND
Package
Class
Interface
Association
Dependency

Figure 3 - Prototype Design

The process of integrating the Bruce application with a PKI using the Façade pattern involved identifying all locations where cryptographic primitives were used in the original source code. Applicable methods were then specified within a façade class to provide links to the Entrust PKI, and the classes that required the use of the cryptographic primitives were adjusted to deal with the façade. The appropriate entities were created within the PKI to allow the notional Bruce administrators to have access to the PKI in order to sign vulnerability checks. Modifications were then made to the way in which output reports were stored and communicated. The changes involved encrypting and signing the output reports.

5. CONCLUSION

This work demonstrates the redesign of a distributed vulnerability scanner to take advantage of the security services offered by a commercial PKI. Designing and implementing distributed applications is complex and using the security services offered by a PKI allows a developer to focus more on the business logic of the application rather than on developing unique solutions to resolve security problems. This revised security services architecture provides enhanced protection for network vulnerability information by using strong standardized PKI mechanisms via a vendor independent interface.

REFERENCES

[BRUCE_DOCS]	Muffet, A., and Watson, K., Bruce source code and documentation, Sun Microsystems, http://www.sun.com/software/communitysource/senss/additional.html
[COPS]	Farmer D. and Spafford E.H., "*The COPS Security Checker System*", Purdue University Technical Report CSD-TR-993, 22 January 1994
[CRACK]	Muffet A., Crack Version 5.0a, available at ftp://coast.cs.purdue.edu/pub/tools/unix/pwdutils/crack
[ECKEL_PAT]	Eckel, B., *Thinking in Patterns with Java, Revision 0.5a*, © 2000 by Bruce Eckel, http://www.bruceeckel.com
[GAMMA]	Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design Patterns, Elements of Reusable Object-Oriented Software*, Addison-Wesley Publishing Company, 1995
[HUMPH]	Humphries, J.W., Carver, C.A. Jr, and Pooch, U.W., "*Secure Mobile Agents for Network Vulnerability Scanning"*, Texas A&M University, June 2000. http://206.96.207.5/curt/
[ISS]	Internet Security Systems Inc. © 1994-2000, *Network and Host-based Vulnerability Assessment*, available from http://www.iss.net/customer_care/resource_center/whitepapers
[JONES]	Jones, A., "*The Challenge of Building Survivable Information-Intensive Systems*", IEEE Computer, August 2000, Volume 33 Number 8 pp. 39-43
[LIC]	SUN COMMUNITY SOURCE LICENSE Version 2.8, (Rev. Date Sept. 16, 1999), electronic copy provided with Bruce download. http://www.sun.com/software/communitysource/senss/additional.html
[NESSUS]	Deraison R., Nessus, © 2000 Renaud Deraison, available at http://www.nessus.org
[SARA]	Todd B., Security Auditor's Research Assistant (SARA), © 94-00 Advanced Research Corporation ® /Updated 18 September 2000, available at http://www-arc.com.sara
[SATAN]	Satan, © Farmer D. and Venema W., 1995, available at http://www.fish.com/satan
[TIGER]	Texas A&M University, Tiger, Tiger, 01 June 1994, available at http://www.net.tamu.edu/ftp/security/TAMU/
[TITAN]	Titan © Archibald M., Farmer D. and Powell B.M., 2000, available at http://www.fish.com/titan
[TRIP]	Tripwire, Inc. © 2001, 18 April 2001, http://www.tripwire.org