Windows Access Control Demystified*

Sudhakar Govindavajhala and Andrew W. Appel
Princeton University
{*sudhakar,appel*}*@cs.princeton.edu*

January 31, 2006

Abstract

In the Secure Internet Programming laboratory at Princeton University, we have been investigating network security management by using logic programming. We developed a rule based framework — Multihost, Multistage, Vulnerability Analysis(MulVAL) — to perform end-to-end, automatic analysis of multi-host, multi-stage attacks on a large network where hosts run different operating systems. The tool finds attack paths where the adversary will have to use one or more than one weaknesses (buffer overflows) in multiple software to attack the network. The MulVAL framework has been demonstrated to be modular, flexible, scalable and efficient [20]. We applied these techniques to perform security analysis of a single host with commonly used software.

We have constructed a logical model of Windows XP access control, in a declarative but executable (Datalog) format. We have built a scanner that reads access-control configuration information from the Windows registry, file system, and service control manager database, and feeds raw configuration data to the model. Therefore we can reason about such things as the existence of privilege-escalation attacks, and indeed we have found several user-to-administrator vulnerabilities caused by misconfigurations of the access-control lists of commercial software from several major vendors. We propose tools such as ours as a vehicle for software developers and system administrators to model and debug the complex interactions of access control on installations under Windows.

1 Introduction

Unix has a simple access control model with three privileges (plus the sometimes mysterious setuid [4]) given to users, groups, and others for operations on just a few kinds of objects (files, directories, etc.). In contrast, Windows attaches access-control lists (prioritized "allow" and "deny" by groups) comprising up to 30 different privileges for operations on about 15 different kinds of objects [6]. For example, on a "service" object one can have the privilege "choose what program.exe is run to effectuate the service." Although

*A *limited* distribution of this paper accompanies a security advisory by USCERT(VU#953860) in December 2005. The tool we built and used to obtain the findings in this paper is not being made publicly available as we are concerned about misuse. We suggest that administrators and developers use the tool SubInACL from Microsoft to investigate individual security descriptors. One could get this tool individually from Microsoft website or obtain it as a part of Windows Resource Kits. We had problems with certain versions of SubInACL tool; we recommend using the version *5.2.3790.1180*. The version 4.0.1.1604 that came with Windows Resource Kit in September 2005 has a bug in the command-line interface which renders the tool useless.

Windows access-control is fine-grained and expressive, unsurprisingly it turns out that ordinary professional software developers at commercial software vendors have difficulty in evaluating the consequences of the access-control configurations that they choose for their software and services. The consequence is that commercial software can and does have privilege-escalation vulnerabilities caused by access-control misconfiguration.

We propose a solution to this problem. We have a logical model of Windows access-control, expressed as inference rules in Datalog which are directly executable in a Prolog system. We have a scanner that reads relevant parts of the Windows registry, file system, and service control manager database on a given host to provide input to our logical model. The model runs, and prints out a list of privilege-escalation vulnerabilities, each one with a trace of how each vulnerability might be exploited.

When we run this on a typical Windows installation managed by a careful systems administrator, we find several exploitable user-to-administrator and guest-to-any-user vulnerabilities caused by misconfigurations in the default installation of software from Adobe, AOL, Macromedia, Microsoft and some anonymous vendors[1].

Our solution, therefore, is that software developers and system administrators evaluate the security of their access-control configurations by running a scanning+analysis tool such as the one we have developed.

In the next section, we give a brief overview of different Windows accounts; in later sections we explain the vulnerabilities found by our tool, the logical model, and the process of gathering configuration data.

2 Windows accounts overview

Windows System Accounts The user mode components of the kernel of the Windows operating system (e.g., *csrss.exe* and *lsass.exe*) run under the *Local System* account. This account has complete access to all the resources of the machine and most of the daemon programs run under this account. In Windows XP, Microsoft introduced the accounts *Local Service* and *Network Service* to run parts of the operating system that do not need complete access to the machine's resources. For example, Windows exports its registry—a global hierarchical database to store data from all programs—over the network. The operating system service responsible for responding to remote registry requests runs as *Local Service*. (Even though *Local Service* does not have complete access to the registry, it can use Windows's delegation facility to access the registry on behalf of the client.) One cannot login into the accounts *Local System*, *Local Service* and *Network Service*, but can control these accounts from any account in the *Administrators* group.

Windows User Accounts The *Administrator* account is the account one uses upon first setting up a workstation or a member server, before creating any other account. It is a member of the *Administrators* group. Any member of the *Administrators* group has complete control of the machine. Windows has various other groups, such as *Authenticated Users*, *Everyone*, *Server Operators*, *Power Users* , *Network Configuration Operators*. Members of the *Power Users* group can create user accounts, can modify and delete accounts they create, stop and start system services which are not started by default. Any principal who supplied a credential to logon belongs to *Authenticated Users* group. Every machine comes with a built-in account *Guest* which is disabled by default. Once enabled, the user *Guest* does not have to supply a credential. Every

[1]The vendors needs time to cope with the issue

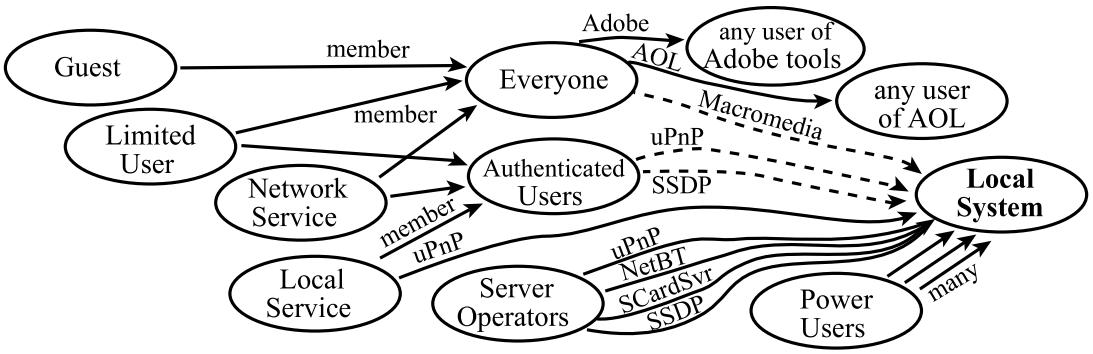

Figure 1: Privilege escalations in software from different vendors. Dashed lines indicate that applying the latest patch fixes the problem.

principal including the *Guest* user is a part of the group *Everyone*. A principal in the *Network Configuration Operators* group can modify the network configuration. The Microsoft documentation for *Server Operators* group reads: *Server Operators can perform most common server administration tasks* [24]. There is also the notion of *Limited Users*, which determines the set of mandatory groups[2] and privileges given to a new user account by the "User Accounts" dialog box when the "Limited" radio-button is selected. In this article we explore privilege escalations between the different user accounts, system accounts, and groups.

3 Security bugs found

We used our tool to see how software is configured by vendors in the default installation, on a professionally managed network of Windows XP client machines with AOL, Adobe and Macromedia software in standard configurations. The results, as shown in figure 1, indicate that unprivileged users can gain administrator privileges through several paths: any arc in the graph is either a group membership (labeled **member**) or a privilege escalation. We now briefly describe Windows *services* and the mistakes developers make in configuring them.

Services Windows does not the support the notion of a setuid bit and developers typically follow a different convention in implementing privileged functionality. A developer would write the program as a *service* and the user sends the command line arguments to the service using a Local Procedure Call [5]. A service can be started automatically or on-demand. Each service has a security descriptor specifying which users are allowed to start, stop and configure the service. Services typically run under the *Local System* account. A daemon program can be implemented as a service that starts automatically.

Several vendors poorly apply the Windows access control model to their services; a common mistake is to assign the SERVICE_CHANGE_CONFIG permission indiscriminately to services. The Windows XP documentation states, "...because this grants the caller the right to change the executable file that the system runs, it should be granted only to administrators" [8]. But that warning fails to explain clearly that permission to configure a service allows both setting the executable *and* selecting the account under which the service runs, e.g., change the "run-as" account to *Local System*[7, 16]. From *Local System*, all things are possible

[2]Each user can belong to more than one group.

(including installing password sniffers to launch further attacks in the guise of any ordinary user).

We now describe specific bugs we discovered; each example is marked with a bullet • and corresponds to one or more labeled arcs in the graph, marked in **boldface**.

• In the default configurations of Windows XP[3] the *SSDP Discovery Service* (**SSDP** in the graph) and the *Universal Plug and Play Device Host service* (**uPnP** in the graph) granted "permission to configure the service" to the *Authenticated Users* group. A normal unprivileged user is a part of the *Authenticated Users* group and hence a normal user can configure the executable and the account under which these services run. Then, the adversary needs to make the service reload the new configuration. He needs to wait for the service to be restarted (he could, for example, force the system administrator to reboot the machine by consuming too many resources so that the system is too slow to respond). We also noticed that usually when a principal is granted the SERVICE_CHANGE_CONFIG permission, he is also granted SERVICE_STOP and SERVICE_START, making it trivial to reload the service in the new configuration, as follows:

```
$ sc config weakService binPath=c:\attack.exe obj=".\LocalSystem" password=""
$ sc stop weakService
$ sc start weakService
```

Via the same SERVICE_CHANGE_CONFIG mechanism, the following (Windows XP Professional) access-control decisions give paths from *Local Service* and *Server Operators* to *Local System*: • The *Local Service* account has permission to configure **uPnP**. • The *Server Operators* group has permission to configure **uPnP**, **SSDP**, *NetBios over TCP/IP* (**NetBT**), and *Smart Card* services (**SCardSvr**). This defeats the principle of least privilege that was the motivation for creating *Local Service* and *Server Operators*. If the adversary were to find a buffer overflow bug in a program running as *Local Service*, this escalation path enables the adversary to take complete control of the host.

Other vendors' software also has access-control configuration bugs in their services: • The *Everyone* group was granted the permission to configure the *Macromedia Licensing Service* (**Macromedia**)[4]. • We also found multiple holes in *service* configuration in other software; we do not discuss the details so that the vendor gets an opportunity to close them. These holes are less serious than the guest-to-administrator and limited-user-to-administrator holes we described earlier.

In addition to Trojan horses via service configuration, some vendors' software is vulnerable to a more traditional kind of file-system-based Trojan-horse vulnerability: • The *Everyone* group has been granted the permission to write to 170 executable (.EXE and .DLL) files from **Adobe**. The adversary can write to these files and wait for a system administrator or other user to execute the files.

Registry The Windows Registry is a global, hierarchical database, where entries are accessed by *keys*. Each registry key has a security context attached to it controlling access to the key. Some registry keys store sensitive information like the path to the executable acting as an user's shell, the library to be loaded by

[3]The vulnerabilities described in this bullet item were open from 2002 until they were closed by Microsoft's patch of August 2004 (though they were still open on the professionally managed network we measured in September 2005); because they are no longer open, we indicate them with dashed lines in the graph. Others that seem to have been patched include, the *Network Service* account had permission to configure the *Distributed Transaction Coordinator* service; *Local Service* had permission to configure *Smart Card Helper* and *Smart Card* services; *Network Configuration Operators* had permission to configure the *DHCP client*, *DNS client* and *NetBios over TCP/IP*. However the patch was not yet installed on the network we measured in September 2005.

[4]This vulnerability was patched by Macromedia in June 2005, hence the dashed line in the figure, but the patch was not yet installed on the network we measured in September 2005.

a program, the identity of an operating system object[5] etc. If an adversary can overwrite the contents of a sensitive key with the path of his library or executable, he could cause his code to be executed[26, 16]. • The standard configuration of **AOL** includes a registry entry binding the name of a DLL file to be loaded and executed (in some circumstances) by the AOL software. The access permissions permit any user to write this entry; the attacker can substitute the name of his own DLL and wait for some other AOL user to execute it. • We also found several weaknesses (potential security holes) in several registry keys from several vendors where an adversary could escalate his privileges; we do not want to present the details before the vendors have a chance to take response measures. If these weaknesses are confirmed security holes, they are less serious than the AOL example presented here. In the AOL example, a guest or limited user could become *Local System*.

Finally, although Microsoft describes *Power Users* as "[i]ncludes many, but not all, privileges of the Administrators group," [3, page 31] it is well known that • there are **many** privilege-escalation paths from *Power Users* to *Local System*; we have found more than 20 with our tool.

As we will explain, we did not have to become experts on AOL, smart card services, and so on to find these vulnerabilities. Our tool simply applies our logic-based analysis to find anomalous accessibilities.

4 Logical model of access-control interactions

We use inference rules in Prolog to express predicates such as, "this *principal* can get that *permission* on that *resource*." By restricting ourselves to a near-Datalog subset of Prolog (i.e., Datalog + bounded-length lists), we guarantee that execution will be finite; in fact, using a Prolog interpreter with *tabled execution* (memorizing of previously derived facts) we will get very efficient execution indeed; our simulations run in under 1 second[6].

Our model has predicates and rules that model the processing of user ids, process tokens and access-control lists on Windows; and other predicates and rules that model how attacks such as Trojan horses can change the access-control state.

A Windows *security identifier* (SID) is a numeric sequence indicating an issuing authority, sub-authorities, and so on; for example S-1-5-21-346327843-89743984-384343-1128. It is used to identify users, groups, built-in accounts like *Local System* etc. A *discretionary access-control list* (DACL) is made up of a list of *access control entries* (ACE) that specify who is allowed what access to an object. An example of an ACE expressed in our model is,

```
ace ( denied,     aceType(['ACCESS_DENIED_ACE_TYPE']),  aceFlags,
      aceMask(['READ_CONTROL', 'SYNCHRONIZE']),    sid('S-1-1-0')).
```

indicating that the the SID S-1-1-0 (the group *Everyone*) is denied the READ_CONTROL and SYNCHRONIZE permissions on the object whose DACL contains this entry.

A resource can be a file, directory, service, or a variety of other kinds of objects. The following example states that a particular file on the host in question has a particular access-control list:

[5]Windows identifies certain operating system objects ("classes") by globally unique identities like *4D36E96B-E325-11CE-BFC1-08002BE10318*

[6]It is well known that the MulVAL framework is scalable and efficient [20].

```
resource(file,  name('C:\\Program Files\\Adobe\\Adobe Help Center\\BIB.dll'),
    acl(owner('S-1-5-21-1268611206-43474576-316617838-59230'),
        dacl( daclFlags,
           acl([ace(aceType(['ACCESS_ALLOWED_ACE_TYPE']), aceFlags,
                    aceMask(['READ_CONTROL', ...]),   sid('S-1-1-0')),
                ace(aceType(['ACCESS_ALLOWED_ACE_TYPE']), aceFlags,
                    aceMask(['READ_CONTROL', ...]),   sid('S-1-5-32-545')),
                ...]))))).
```

(ellipses not in original). One of the inputs to our model is a long list of facts like this, gathered by our scanner from the Windows file system, services and registry. We refer the reader the to a previously published technical report for the details of formal modeling of Windows access control semantics [13].

We use the logic of access-control to calculate derived predicates such as, canAccess(*Principal*, *Permission*, *Resource*), meaning that a certain SID, *Principal*, can get a given *Permission* on a given *Resource*. (In Prolog, capital letters start variable names and small letters start constants, constructors, and predicates.) In fact, Windows does this in two stages: from an SID it constructs a *process token* that contains information about all the group memberships of the account; then each access-control check is done by comparing the DACL to the process token.

Modeling attacks. To reason about attacks, we have rules such as the following.

```
canAccess (Principal, Permission, Resource) :-
     member (Principal, Group), canAccess (Group, Permission, Resource).
```

If *Principal* belongs to *Group*, he gets any permission that the group can get (we oversimplify here by leaving out process tokens). In Prolog, the conclusion is written first and the premises are after the `:-` symbol.

```
compromised (guest).

canAccess (OtherPrincipal, Permission, Resource) :-
     compromised (Principal),  canAccess (Principal, write_dac, Resource).
```

The rule above says, if you have the WRITE_DAC ("write discretionary access-control list") permission on the resource, you can make any *OtherPrincipal* have any *Permission* on the resource. But you have to "want" to do it—you would "want" to do this only if the attacker had already compromised your account and you are (unknowingly) executing the attacker's program. We assume the `guest` user is initially *compromised,* i.e., "acts on behalf of an attacker." In many circumstances, it is also reasonable to assume `compromised(limited_user)`, that some ordinary unprivileged (but authenticated) user is compromised.

So, if you have WRITE_DAC and you're compromised, then you can give any permission to any other principal (including yourself) for that resource—including, of course, SERVICE_CHANGE_CONFIG.

```
compromised (Principal) :-
     compromised (Attacker),
     canAccess (Attacker, service_change_config, Service).
```

If you have been compromised by the attacker, and your account can change the service configuration of a *Service* to run the attacker's .EXE file in a process owned by *Principal*, then you can compromise that *Principal* (to simplify the presentation, we omit the need to start the service).

```
compromised (Principal) :-
     compromised (Attacker),
     canAccess (Attacker, write, File),
     canAccess (Principal, execute, File).
```

If you can write to a file that some *Principal* can execute, then you can install a Trojan Horse that makes that *Principal* act on your behalf (a more refined model considers the likelihood that the victim will actually execute your file, with commonly used executables in a public directory considered more likely executed).

The query. An example of a query we can give to the Prolog engine is, "tell me all possible accesses, legitimate and otherwise, by principals to resources," but this delivers far too many results, most of which are harmless. A more interesting query is, "can the guest user compromise an administrator?"

```
?-  compromised(localSystem).
```

Queries modeled on this one calculate the vulnerability graph.

5 Scanner

Our scanner is very simple: it reads the file system (directories and their access control lists) and registry (services and their ACLs), and produces data such as the `resource(file,...)` datum shown in the previous section. In addition, the scanner gathers the ACL of any registry item whose contents is a name ending in .exe, .dll, .bat, and so on, which are presumed to be executables through which Trojan Horses might be installed. Our scanner takes about 5 minutes to run on a typical host.

Our scanner can run with almost no privileges (i.e., as a Limited user) and still gather data sufficient to produce the graph in Section 1. This is desirable, because a system administrator might legitimately worry about running yet another high-privilege software tool that might harm his system. It also means, of course, that all that raw information was already available to sophisticated attackers. We discovered only one weakness — one not discussed in this report — that can be discovered only with administrator privileges. This weakness is just a weakness and cannot result in a direct attack unless one can attack certain cryptographic mechanism — meaning a fundamentally new technique is required by the adversary.

6 Related work and conclusion

Many application programs demand too many privileges, more than strictly necessary to access the data on which they operate. Chen *et al.* [5] explain why this is harmful (users demand and receive the privileges

necessary to run the applications, which then gives them privileges to do harm) and have built an analysis tool for finding such situations. That is the inverse of what we have described in this paper, which is applications granting too many privileges, but we believe our detailed model of Windows access-control could also be used to do "least-privilege-incompatibility" analysis.

Modeling vulnerabilities and their interactions can be dated back to the Kuang and COPS security analyzers for Unix [2, 11]. Recent works in this area include the one by Ramakrishnan and Sekar [21], and the one by Fithen et al [12]. A major difference between our work and these works is the application to Windows platform. There is a long line of work on network vulnerability analysis [28, 25, 22, 23, 1, 19]. These works did not address how to automatically integrate vulnerability specifications from the bug-reporting community into the reasoning model. Ritchey and Amman proposed using model checking for network vulnerability analysis [22]. Sheyner, et. al extensively studied attack-graph generation based on model-checking techniques [23]. These approaches suffer from problems of state space explosion. While it is foreseeable that these approaches can be made to work, it has not been demonstrated that the approach scales for large networks. Amman et. al proposed a graph-based search algorithms to conduct network vulnerability analysis [1]. This algorithm is adopted in Topological Vulnerability Analysis (TVA) [14], a framework that combines an exploit knowledge base with a remote network vulnerability scanner to analyze exploit sequences leading to attack goals. However, it seems building the exploit model involves manual construction, limiting the tool's use in practice. Intrusion detection systems have been widely deployed in networks and extensively studied in the literature [9, 18, 15]. Unlike IDS, MulVAL aims at detecting potential attack paths *before* an attack happens.

The MulVAL framework understands the semantics of two disparate operating systems — Windows and Linux. We are not aware of prior work that can understand the semantics of disparate operating systems. It has not been convincingly established that prior approaches are actually useful in practice. In contrast, we applied the MulVAL framework to find bugs in professionally managed networks and configurations of commercial software. The MulVAL framework is about analyzing to see if an adversary can combine one or more buffer overflow bugs in software to compromise his target. The results presented in this paper represent a special case application of the MulVAL framework. These results followed naturally once we asked the question: *"Is it possible that even in the absence of buffer overflow bugs, the adversary could use buggy configurations to attack the target?"* We did not have to change our framework to obtain the results described in this report. In contrast to prior work, we are convinced that that the MulVAl framework is end-to-end, modular, automatic, efficient, flexible and scalable.

Previous work has shown that static configuration scanning can be useful: OVAL [27] finds software on your machine that is the subject of a security advisory (e.g., CERT has found a buffer-overrun vulnerability). Other work has shown that Datalog can be expressive and effective in reasoning about access control [10] and in reasoning about propagation of attacks across multi-host networks [20].

Microsoft's *Trustworthy Computing Security Development Lifecycle* project [17] addresses many of the issues discussed in this paper, which explains why the vulnerabilities we describe in Microsoft's own software were closed in mid-2004.

We also believe that buggy access-control configuration is a problem that could affect vendors and operating system versions not discussed in this paper. There is no reason to believe that only developers at Adobe, AOL, Macromedia and Microsoft made mistakes in access-control configuration. Our views are vindicated when we found similar bugs in a commonly used product from a different vendor very close to preparing the

final version of this document[7]. Even if the operating system is secure at installation time, it is worrisome that it could become insecure upon installing commonly used (required) software.

Why bugs? We believe there are two reasons that explain these bugs. The developers did not completely understand the (complex) semantics of the operating system. The operating system documentation is slightly unclear (misleading enough that a casual programmer makes a mistake). For example, a developer could be wanting to give an untrusted user the permission to *start* or *stop* a service. He decides to give *all access* to the user; which includes the permissions to *write the security descriptor* and *configure the service*.

Our contribution here is to show that a Datalog model of Windows XP access control is surprisingly useful even on a single-host system. We believe the reason is that Windows access-control is sufficiently complex that programmers and administrators have difficulty understanding and debugging their access-control decisions without tools.

Acknowledgments. Discussions with Varugis Kurien, Adrian Oney, Brandon Baker, and Scott Field helped clarify our understanding of the Windows security model. We thank Wayne Boyer, Ed Felten, Xiaolan Zhang, Alex Halderman, John Lambert, Miles McQueen, Michael Steiner, Ruoming Pang, Daniel Dantas, Xinming Ou, and Matt Thomlinson for helpful comments on earlier drafts of this paper. An internship at Microsoft by the first author in the summer of 2005 was a valuable opportunity to gain experience with the APIs used in access control and in querying the Windows registry.

References

[1] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of 9th ACM Conference on Computer and Communications Security*, Washington, DC, November 2002.

[2] R. Baldwin. Rule based analysis of computer security. Technical Report TR-401, MIT LCS Lab, 1988.

[3] Ed Bott and Carl Siechert. *Microsoft Windows Security Inside Out: for Windows XP and Windows 2000*. Microsoft Press, 2003.

[4] Hao Chen, David Wagner, and Drew Dean. Setuid demystified. In *Proceedings of the 11th USENIX Security Symposium*, pages 171–190, Berkeley, CA, USA, 2002. USENIX Association.

[5] Shuo Chen, John Dunagan, Chad Verbowski, and Yi-Min Wang. A black-box tracing technique to identify causes of least-privilege incompatibilities. In *Proceedings of Network and Distributed System Security Symposium, 2005*, February 2005.

[6] Microsoft Corporation. Access rights and access masks. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/secauthz/security/access_rights_and_access_masks.asp, October 2005. web page fetched October 9, 2005.

[7] Microsoft Corporation. ChangeServiceConfig. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/changeserviceconfig.asp, 2005.

[7] We are not presenting the details so that the vendor gets an opportunity to take appropriate response measures.

[8] Microsoft Corporation. Service security and access rights. http://windowssdk.msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/service_security_and_access_rights.asp, October 2005. web page fetched October 9, 2005.

[9] Frdric Cuppens and Alexandre Mige. Alert correlation in a cooperative intrusion detection framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, page 202. IEEE Computer Society, 2002.

[10] John DeTreville. Binder, a logic-based security language. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, page 105. IEEE Computer Society, 2002.

[11] Daniel Farmer and Eugene H. Spafford. The cops security checker system. Technical Report CSD-TR-993, Purdue University, September 1991.

[12] William L. Fithen, Shawn V. Hernan, Paul F. O'Rourke, and David A. Shinberg. Formal modeling of vulnerabilities. *Bell Labs technical journal*, 8(4):173–186, 2004.

[13] Sudhakar Govindavajhala. Status of the MulVAL project. Technical Report TR-743-06, Department of Computer Science, Princeton University, 2006.

[14] Sushil Jajodia, Steven Noel, and Brian O'Berry. Topological analysis of network attack vulnerabity. In V. Kumar, J. Srivastava, and A. Lazarevic, editors, *Managing Cyber Threats: Issues, Approaches and Challanges*, chapter 5. Kluwer Academic Publisher, 2003.

[15] Samuel T. King, Z. Morley Mao, Dominic G. Lucchetti, and Peter M. Chen. Enriching intrusion alerts through multi-host causality. In *The 12th Annual Network and Distributed System Security Symposium (NDSS 05)*, Feb. 2005.

[16] John Lambert, Matt Thomlinson, and Vishal Kumar. Microsoft Corporation, personal communication, July 2005.

[17] Steven B. Lipner. The trustworthy computing security development lifecycle. In *20th Annual Computer Security Applications Conference (ACSAC 2004)*, pages 2–13, December 2004. See also msdn.microsoft.com/security/sdl.

[18] Peng Ning, Yun Cui, and Douglas S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 245–254. ACM Press, 2002.

[19] Steven Noel, Sushil Jajodia, Brian O'Berry, and Michael Jacobs. Efficient minimum-cost network hardening via exploit dependency graphs. In *19th Annual Computer Security Applications Conference (ACSAC)*, December 2003.

[20] Xinming Ou, Sudhakar Govindavajhala, and Andrew W. Appel. Mulval: A logic-based network security analyzer. In *14th USENIX Security Symposium*, 2005.

[21] C. R. Ramakrishnan and R. Sekar. Model-based analysis of configuration vulnerabilities. *Journal of Computer Security*, 10(1-2):189–209, 2002.

[22] Ronald W. Ritchey and Paul Ammann. Using model checking to analyze network vulnerabilities. In *2000 IEEE Symposium on Security and Privacy*, pages 156–165, 2000.

[23] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 254–265, 2002.

[24] William R. Stanek. Windows 2000 server: Using default group accounts. http://www.microsoft.com/technet/prodtechnol/windows2000serv/evaluate/featfunc/07w2kadc.mspx. web page fetched January 28, 2006.

[25] Steven J. Templeton and Karl Levitt. A requires/provides model for computer attacks. In *Proceedings of the 2000 workshop on New security paradigms*, pages 31–38. ACM Press, 2000.

[26] Yi-Min Wang, Roussi Roussev, Chad Verbowski, Aaron Johnson, Ming-Wei Wu, Yennun Huang, and Sy-Yen Kuo. Gatekeeper: Monitoring auto-start extensibility points (ASEPs) for spyware management. In *Usenix LISA: 18th Large Installation System Administration Conference*, November 2004.

[27] Matthew Wojcik, Tiffany Bergeron, Todd Wittbold, and Robert Roberge. Introduction to OVAL: A new language to determine the presence of software vulnerabilities. http://oval.mitre.org/documents/docs-03/intro/intro.html, November 2003. Web page fetched on October 28, 2004.

[28] Dan Zerkle and Karl Levitt. NetKuang–A multi-host configuration vulnerability checker. In *Proc. of the 6th USENIX Security Symposium*, pages 195–201, San Jose, California, 1996.