A Software Fault Tree Approach to Requirements Analysis of an Intrusion Detection System

Guy Helmer, Johnny Wong, Mark Slagell, Vasant Honavar, Les Miller*
Department of Computer Science
Iowa State University
Robyn Lutz†
Iowa State University & Jet Propulsion Laboratory
{ghelmer,wong,rlutz,slagell,honavar,lmiller}@cs.iastate.edu

October 16, 2000

Abstract

The use of software fault trees for requirements identification and analysis in an Intrusion Detection System (IDS) is described. Intrusions are divided into seven stages, following Ruiu, and a fault subtree is developed to model each stage (e.g., reconnaissance, penetration, etc.). The software fault tree approach supports requirements evolution as new intrusions are identified as well as prioritized, incremental development of the distributed IDS. The IDS under development is a collection of mobile agents that detect, classify, and correlate system and network activities. Analysis of the software fault trees and the minimum cut sets identify the software requirements. These derived requirements include what activities are to be monitored in the agent software, what intrusion characteristics the agents should correlate, where the IDS agents are to be placed to feasibly detect the intrusions, and what countermeasures the software should take. Two examples of intrusions demonstrate how the staged, software fault tree approach has been used in development of the IDS.

1 Introduction

Software Fault Tree Analysis (SFTA) is a method for identifying and documenting the combinations of lower-level software events that allow a top-level event (or root node) to occur. When the root node is a hazard, the SFTA assists in the requirements process by describing the ways in which the system can reach that unsafe state [15]. The safety requirements for the system can then be derived from the software fault tree, either indirectly [3][17] or directly via a shared model [9].

In the work described here, we use SFTA to assist in determining and verifying the requirements for an intrusion detection system. The root node of the top-level SFTA is not strictly a hazard, as in a safety analysis, but an intrusion. An intrusion is a violation of a system's security policy. Intrusions result in compromise of exclusivity (unauthorized disclosure of data or use of services), integrity (unauthorized modification of data), or availability (denial of service). Whereas safety failures are often accidental or unexpected, intrusions are intentional, perpetrated by individuals, and can be expected to occur. Both safety and security failures represent potentially significant or catastrophic losses.

*Funded in part by the Department of Defense.

†Some of this work was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with NASA. Funded in part by NASA's Code Q Software Program Center Initiative, UPN 323-08.

Intrusions can occur in a variety of ways. The software fault tree models the combinations and sequences of events by which intrusions can occur. The understanding and capture of domain knowledge needed to accurately define the requirements on an Intrusion Detection System (IDS) is difficult. Questions such as what intrusions can be feasibly detected by the IDS software, at what stage of an intrusion the IDS software should detect each attack, and what assurances can be given that the IDS software detects intrusions must be addressed by the requirements analysis. The goal is not to build a system in which the root node never occurs, but to build an IDS in which the root node never occurs undetected.

We are investigating the formal underpinnings of intrusion detection systems (IDS's) in terms of how to describe intrusions, identify intrusion characteristics and provably detect intrusions based on observable characteristics. Existing intrusion detection systems tend to be built by selecting a set of data sources and developing a classification system to identify some set of intrusions using the selected data [11]. A broader approach that begins with a thorough analysis of intrusions and supports development of a theoretical model of intrusion detection will better help answer questions about which intrusions are detectable, how they can be detected, how the data from different sensors should be correlated, and to what extent we can be assured that a report of an intrusion or a non-intrusion is accurate.

The primary contribution of the work described here is to analyze the intrusion domain using software fault trees in order to determine the requirements for an IDS. The SFTA models the stages of intrusion in a structure that supports discovery and reasoning about requirements. In addition, the SFTA supports requirements evolution. The fault tree can be updated as new intrusions are identified, an essential feature for security applications. The SFTA also allows incremental development of the IDS as progressively more paths to the root node (or to the root node of a subtree) are blocked by the software. Inspection of the minimum cut sets provides guidance as to where software monitors in the IDS should be required. Finally, path coverage metrics provide some verification that the IDS requirements are correct.

The rest of the paper is organized as follows. Section 2 provides some background on SFTA and graph-based IDS. Section 3 describes fault tree modeling of intrusions. Section 4 elaborates two specific examples from our experience with SFTA for the requirements determination of an IDS. Section 5 discusses the results of the use of SFTA in terms of requirements identification and analysis, requirements evolution, and verification. Section 6 contains concluding remarks.

2 Background

2.1 Software Fault Tree Analysis

The Software Fault Tree Analysis used to model intrusions is a backward search. It begins with a known hazard (here, an intrusion) as the root node and traces back through the possible parallel and serial combinations of events that caused such an intrusion. Due to space limitations, the reader is referred to [15][20] for more precise descriptions of fault trees. The Saphire software from the Idaho National Engineering and Environmental Lab was used to draw and edit the fault trees.

A *cut set* is a set of basic events whose occurrence causes the system to fail [20]. A cut set is called a *minimum cut set* if it cannot be reduced and can still cause the system to fail [20]. A minimum cut of a fault tree gives a minimum set of successful events necessary to satisfy the root. Manian, et al. [18] use Binary Decision Diagrams as an alternative to cutset-based solutions of fault trees for large, combinatorial solutions. In our current work, the size of the fault trees has been manageable using traditional cutset-based solutions. A minimum cut of an intrusion fault tree describes a scenario of a use case in which an attacker successfully exploits security flaws to achieve the goal of compromising the system.

2.2 Graph-Based Models of Intrusion Detection Systems

Several graph-based modeling techniques for IDS exist, but they model the intrusion detection system rather than the intrusion itself. For example, GrIDS, the Graph-based Intrusion Detection System, detects misuse in a system by dynamically building graphs that model the communication activities in a network [22]. The graph depends on user-defined rules to identify suspicious patterns and models intrusion detection, rather than intrusions. ARMD, the Adaptable Real-time Misuse Detection system, represents misuse signatures as directed acyclic graphs [16]. Unlike the object/event model used by GrIDS, the graphs are not amenable to aggregation. IDIOT, Intrusion Detection In Our Time, is an IDS that uses a custom language and a variant of CPNs for misuse detection [14][13].

3 Developing Fault Trees for Intrusions

Intrusion fault tree modeling draws from a variety of sources. The standards used in current TCP/IP networks are publicly available. Proposals and standards for IP networks are published by the Internet Engineering Task Force (IETF) as Requests for Comment (RFC's) and Standards (STD's). Implementations of most network protocols are freely available in software such as Linux, FreeBSD, and Apache, allowing public review for security issues. Numerous researchers and hackers actively discover and publish security vulnerabilities in public forums including mail lists such as `bugtraq` and web sites such as `www.securityfocus.com`. Jansen et al., [12] discuss the possible roles of mobile agents in effective intrusion detection.

Faults that are generally UNIX-centric are considered in the fault trees, although many similar problems (e.g. buffer overflows) exist in software on other systems. Rather than looking directly at the source code for these systems, the immense body of publicly-discussed vulnerability information is used as input for development of the sample fault trees discussed here.

3.1 Reasonable Fault Trees

Each complete, successful intrusion can vary greatly from all other intrusions, and attempts to analyze complete intrusions are difficult. A monolithic fault tree that would attempt to describe all attacks would be huge, unwieldy, and less useful than several trees divided in a systematic manner. A reasonable approach is to divide intrusions into stages of attacks that achieve intermediate goals of the attacker, and to develop fault trees that model each of the stages.

Initially, four stages of intrusions were identified (reconnaissance, penetration, embedding, and operations). Ruiu's analysis of intrusions [21] separates vulnerability identification from the reconnaissance stage, separates the control stage from the penetration stage, and divides the operations stage into data extraction & modification and attack relay stages. Independent analysis of documented intrusions [23] and hypothesized intrusions agree with Ruiu's more complete breakdown: (1) Reconnaissance, (2) Vulnerability Identification, (3) Penetration, (4) Control, (5) Embedding, (6) Data Extraction & Modification, and (7) Attack Relay.

3.2 Developed Examples of SFT for Intrusions

The OR gates in the fault trees shown are "true" if any input is true. The AND gates are "true" if all inputs are true in the current context, where the context may be a virtual network connection, a user's login session, a series of related transactions, or some other temporal context. The AND gates in the fault trees shown generally assume that the child events occur in left-to-right order. (Hansen et al. [9] discuss the ambiguities of traditionally accepted fault trees.)

The sample fault trees are not complete, even for known intrusions, but represent a significant subset of the intrusions of most concern to administrators of distributed networks. An example of how paths in the trees can describe a complete intrusion is discussed below.

We consider an *active* attack to be an attack that includes events that can be seen within the distributed system under the organization's control. The majority of attacks in an intrusion are active. We consider a *passive* attack to be an attack that leaves no trace within an organization's distributed system. Examples of passive attacks are password sniffing or DNS zone transfers from an off-site secondary DNS server.

3.2.1 Reconnaissance

The reconnaissance phase identifies potential targets within an organization's networks. Network targets include not only multiuser hosts (e.g., UNIX or Windows/NT systems) but also routers, intelligent hubs, and perhaps even modems. The services offered by systems and names of users on the systems are also useful information for an attacker. Figure 1 shows a sample fault tree for the reconnaissance phase.

3.2.2 Vulnerability Identification

Vulnerability identification is closely related to reconnaissance. In this phase, an attacker searches for vulnerabilities that can lead to penetration. The attacker sequentially scans many ports looking for versions of remote control services known to be vulnerable to attack (e.g., BackOffice or NetBus). Port scanning is a "noisy" active attack, and is usually easy to detect unless done very slowly. The software fault tree in Figure 2 models this vulnerabililty identification phase of the intrusion.

3.2.3 Penetration

Penetration occurs when an attacker obtains unauthorized access to a system. Penetration methods include exploitation of various network server daemon vulnerabilities (poor authentication and buffer overflows), authenticating with illicitly obtained passwords, and TCP session hijacking. Figures 3, 4, 5, and 6 together represent a sample fault tree for the penetration stage of intrusions.

3.2.4 Control

An attacker needs to gain sufficient privilege in a system to continue to the next stages of the intrusion. Often an attacker must obtain privileges equivalent to those of the system administrator to gain sufficient control of a system. If the penetration was particularly effective and sufficient privilege was already gained, this step may not be necessary.

Mechanisms traced in Figure 7 include exploiting buffer overflows in privileged local programs, exploiting race conditions in temporary files or signals, exploiting weak permissions on critical files and devices, and cracking a password for an administrator's account.

3.2.5 Embedding

Embedding involves the installation or modification of a system so that even if the attacker is discovered and steps are taken to recover the system, the attacker will still be able to enter the system. For example, the system bootstrap code could be modified to re-insert backdoors if the system executable programs are restored from backups or installation media. Typical embedding techniques include installing Trojan horses, backdoor, and other rootkit programs, removing traces of the intrusion from system logs, and disabling detection systems.

Figure 8 identifies two different rootkit installations by matching particular sets of modified system files. A rootkit is a collection of embedding programs that allow an attacker to hide his activities and may include programs for use in the next step, data extraction & modification.

3.2.6 Data Extraction & Modification

In the data extraction and modification phase, the attacker gathers information about the configuration and operation of the system. Covert channels may be used to move discovered data from the compromised system to the attacker's base.

3.2.7 Attack Relay

After a system is fully compromised, it may be used for attack relaying. Attacks can be launched against affiliated (trusting) hosts to expand the number of hosts under the attacker's control. A system also may simply be used to participate in distributed denial-of-service attacks [8][6][7][5]. Figure 9 represents some basic faults seen from Stacheldraht, Tribe Flood Network, and Trinoo distributed denial of service attacks. Many other forms of attack relaying exist, including automated and manual means.

4 Experience with Fault Trees for Intrusions

The relationship of the developed fault trees to two actual intrusions is examined in this section. Each intrusion follows one of the multiple paths through each of the staged subtrees in Figs. 1-9. A portion of the fault tree of Figure 6, describing the FTP bounce attack, was selected for further analysis. The FTP bounce attack subtree is particularly interesting because it involves several time-ordered steps which must take place for the attack to be successful.

4.1 Example 1: FTP SITE EXEC Intrusion

The FTP SITE EXEC attack against the wuftpd daemon [2] is a buffer overflow exploit. When someone logs into the wuftpd daemon as the user `anonymous` or `ftp`, the daemon requests that the email address be entered as the password. However, an attacker can instead send malicious shell code in response to the password prompt. Then, if the SITE EXEC command is enabled, the attacker can send a SITE EXEC command with %-formatting characters that cause a buffer to overflow with data previously obtained as the password.

In the reconnaissance stage of the attack, an intruder discovers an anonymous FTP server host by using any one or more of the methods under the "HostDiscovery" node in the reconnaissance tree in Figure 1.

The intruder also discovers the availability of the FTP server by one of the methods under the "TCP-Service-Discovery" node in the reconnaissance tree.

In the next stage of the attack, the intruder identifies a vulnerability (a path through the subtree in Fig. 2). The intruder may or may not take the time to make a connection to the FTP server and verify that the version number reported by the server is vulnerable to the FTP SITE EXEC attack. (Known FTP vulnerabilities are not expanded under the "FTPD" node in the vulnerability identification tree in Figure 2.)

Fig. 4 shows the penetration fault tree as this intrusion scenario continues. The intruder can now connect to the FTP server, give "anonymous" or "ftp" as the user name, and enter malicious shell code as the password. The intruder then issues a SITE EXEC command containing printf-style substitution character sequences. This is an attempt to overflow the character buffer on the process' stack with the data from the previously-entered "password." If the overflow is successful, the code provided by the intruder is executed with root privileges. A successful FTP SITE EXEC attack also gives the attacker control, so the attacker can move on to the later stages of the intrusion. (In this intrusion scenario, we assume the successful penetration results in privileged access, so the control phase of the intrusion may be by-passed.)

In the embedding stage, the intruder can install the Linux Rootkit version 4, which replaces a number of programs with Trojaned implementations that hide the attackers activities. Figure 8 describes the changes to the file system that result from the installation of the Linux Rootkit version 4.

In the data extraction stage, the intruder installs and runs a password sniffer that takes user names and passwords from telnet and ftp sessions on the LAN.

In the final stage of the intrusion, shown as a path through the attack fault tree in Fig. 9, the intruder installs and runs a distributed denial of service agent, such as Trinoo, TFN, or Stacheldraht. The intruder can then use the system to execute attacks against other networked sites.

4.1.1 Derived IDS Requirements

The software fault trees involved in this intrusion helped identify the software requirements for the mobile agent software tasked with detecting the FTP SITE EXEC intrusion. Examination of the trees shows that in the penetration subtree the path taken by the FTP SITE EXEC attack, it is feasible to detect this penetration in software. An intrusion detection system should thus monitor PASS commands in an FTP session for data that does not represent a valid sequence of printable characters. That is, an invalid sequence of printable characters is a minimum cut event. The analysis does not say anything about how the monitoring should be implemented or performed; it merely leads to requirements for the intrusion detection system.

4.2 Example 2: FTP Bounce Intrusion

The FTP bounce attack can be used to transfer data to a network port to which an attacker does not normally have access [1]. One way to exploit this problem is to send data to a remote shell server that trusts the FTP host via the FTP server. After an attacker discovers an FTP server and a host running rsh that might trust the FTP server, the attacker tries this exploit:

1. Uploads a specially-formatted file to the FTP server;
2. Issues an FTP PORT command that directs the FTP server to send its next download to port 514 on the target host;
3. Issues an FTP GET command to "download" the contents of the previously-uploaded file into port 514 on the target; the GET command opens a connection from the FTP server on port 20 to the rsh daemon on the target.
4. If the target trusts the FTP server, the rsh daemon will accept the contents of the file as if it were user input and execute the given command.

The following steps in an intrusion based on an FTP bounce attack show how the trees fit the entire intrusion.

In the Reconnaissance stage of the intrusion, the intruder discovers an FTP server host and a target host, using any one or more of the methods under the "HostDiscovery" node in the reconnaissance tree. The intruder also discovers the availability of the FTP server and RSH server by one of the methods under the "TCP-Service-Discovery" node in the reconnaissance tree.

As in the previously discussed intrusion, during the Vulnerability Identification stage, the intruder may or may not take the time to make a connection to the FTP server and verify that the version number reported by the server is vulnerable to the FTP bounce attack. In this path through the subtree, the intruder also needs a directory on the FTP server to which he may upload a file; if the intruder has no access to the FTP server other than "anonymous", the intruder will have to search for such a directory. The large number of known FTP vulnerabilities has not yet been expanded under the "FTPD" node in the vulnerability identification tree.

The intruder will likely have to assume that the target host trusts the FTP server host, unless the intruder already has some access to the target host and can read the `/etc/hosts.equiv` or `~root/.rhosts` files. We have not considered "insider access" in the vulnerability identification tree.

The intrusion continues through the penetration subtree, with the intruder uploading the shell command file to the FTP server and issuing the appropriate FTP commands to cause the FTP server to "download" the file into the target's RSH service. The "FTP-Bounce" subtree of Fig. 6 shows the required FTP commands and responses. The structure of the subtree enforces the order of the events in the FTP command/response stream.

The successful FTP bounce attack mounted against a privileged account on the target also gives the attacker control, so the attacker can move on to the later stages of the intrusion. These later stages are omitted here for reasons of space since the path at this point is identical to that of the previously described intrusion.

4.2.1 Derived IDS Requirements

Analysis of the subtree concerning the FTP bounce attack shows that, in order to detect the attack, software requirements on the IDS are to monitor commands and responses in an FTP session, to monitor rsh connections, and to correlate outputs from the two monitors to determine whether an FTP bounce attack was attempted and whether the attack was successful. As before, the analysis does not say anything about how the monitoring should be implemented or performed, but merely yields requirements for the intrusion detection system.

4.2.2 Countermeasures for the FTP Bounce Attack

Each of the steps in the intrusion detailed above is part of a scenario which fits a minimum cut of the corresponding fault tree. Inspecting the minimum cuts for each intrusion leads us to the best point at which to apply countermeasures. Countermeasures in intrusion detection systems typically include alerts to the system manager (via email, paging, or simply log messages), termination of network connections or logins, and disabling user accounts.

We examined the minimum cut from the penetration tree for the FTP Bounce Attack and informally considered the cost of applying countermeasures at each node. Cost included the complexity of the software required and the effect on the legitimate users of the system. It appears that the lowest cost countermeasure would be killing the TCP connection made from the FTP server to the RSH server; countermeasures at other nodes would either be prohibitive to implement, prevent legitimate uses of the FTP or RSH services, or be too late to terminate the FTP bounce intrusion.

5 Discussion of Results

Software fault trees for intrusions explore the necessary and sufficient combinations of events that lead to exploitation of a vulnerability. Development of fault trees for intrusions enabled a variety of discovery and verification activities. We summarize these briefly here and refer the reader to the previous section for examples.

Requirements Identification & Analysis Fault trees document properties of intrusions and allow for analysis of intrusion properties.

Domain Understanding and Documentation. Capturing this domain understanding is frequently difficult in the security arena. Software fault trees provide a standard, easy-to-use format for documentating properties of intrusions by system and network experts.

Determining Requirements. Each minimum cut models an intrusion sequence that the software may be required to recognize. Identification of leaf events in the fault tree illustrates what components of a distributed system must be monitored to detect the intrusion. In addition, analysis of intrusion fault trees exposes conditions where countermeasures may be successfully applied by an intrusion detection system to intervene before the intrusion is successful.

Fault Detectability Analysis. This refers to the ability of the system to detect the problem if it appears during system operation [4]. Determining which characteristics of intrusions can be monitored is an essential part of the requirements analysis for an IDS. For example, there exist certain intrusive events which do not have any discernible effect in a site's distributed system. Such events include "DNS zone transfers" from off-site secondary name servers and passive password sniffing. Marking these events appropriately in the fault tree allows analysis of which intrusions would be particularly difficult to detect, and may give hints regarding ways to prevent such intrusions from occurring.

Requirements Evolution & Incremental Development Software fault trees support intrusion detection system development and maintenance activities.

Prioritization of Requirements. The addition of historical information regarding likelihood and severity on the leaf nodes (not addressed in this paper) would assist in prioritizing requirements [15]. In addition, based on severity and likelihood information from the fault trees, alert priorities could be encoded in an intrusion detection system.

New attacks. Newly-discovered attacks need to be integrated into the intrusion fault tree. Such new information may encourage re-organization of the fault tree, as when a new attack depends on a set of circumstances that is already diagrammed in the fault tree, or the addition of a subtree (either new or reused). The changes necessary in the intrusion fault tree to incorporate information about newly-discovered attacks will then guide the necessary modifications of the intrusion detection requirements and design to detect the new attacks.

Verification Once confidence is established in the software fault tree, primarily through expert review, the design of the intrusion detection can then be traced to the software fault tree to determine its completeness and correctness.

Based on the testing strategy of Puketza et al. [19] the SFTA can be used to test the design and implementation of an IDS. Given a subtree of an SFTA that describes related intrusive events, define the subtree to be an equivalence class for the set of intrusions. Select one or more representative minimum cuts of the subtree to be tested. Then, given scenarios which are positive and negative examples of the intrusions, execute the intrusions and determine whether the subtree accurately matches the events. The scenarios form a set of representative test cases for the equivalence class.

We do not interpret the fault tree directly as requirements, unlike [9], where the fault tree has a formal semantics. A less formal approach was desired in the intrusion application because we want the fault tree to be developed and maintained by system support personnel rather than by experts in formal specification. It is primarily the support personnel's knowledge of the system and its vulnerabilities that the fault tree is intended to capture. To understand this, a brief description of the larger IDS system is in order.

The intrusion fault tree work described here is the requirements phase of a larger effort to provide a more formal framework for building IDS. The IDS will use mobile agents in a distributed system to collect audit data, classify it, correlate information from the different mobile agents, and detect intrusions. The intrusion

fault tree drives the requirements for these mobile agents and the intrusion detection system. The fault tree is mapped, by a correctness-preserving transformation, into colored Petri nets (CPNs) that serve as the design specification of the mobile agents in the IDS. Interactive simulation of these CPNs gives additional verification that the design satisfies the requirements (i.e., blocks the relevant path(s) in the intrusion fault tree). Code for the IDS mobile agents is generated from the CPNs and tested using, among other scenarios, the minimum cuts through the intrusion fault tree. Currently, prototypes exist of each of these phases (i.e., some CPNs and some mobile agents for some intrusions) with work on-going to partially automate the code generation.

Two interesting aspects of the requirements phase of this work are as follows. First, at least for the prototypes, there has been no requirements specification. Instead, the intrusion fault trees have been interpreted as specifications of the combinations of events that must be detected. That is, the IDS requirements are that each of the intrusion sequences possible in the fault tree should be detected as soon (low in the tree) as possible. The leaf events describe what components of a distributed system must be monitored by the mobile agent software. No separate requirements specification document has been developed.

Second, the intrusion fault trees have had to be extended with additional information specific to a particular system prior to their mapping into CPNs. This information is of three types: Trust (which other members of a distributed system each member trusts); Context (which events must all involve the same host(s) or connection(s), process(es) or session(s)); and Temporal orderings (e.g., which events must be adjacent with no intervening events, or follow within a specific interval of time). Without this additional system-specific information, the IDS yields many false positives, detecting intrusions where, in a specific network, there is none. That is, the set of events marked as intrusions by the fault tree is a superset of the set of events that are actually intrusions in any specific network and must be constrained by additional network-specific knowledge. More information on these constraints is available in [10].

6 Summary and Future Work

The use of software fault tree analysis to model intrusions to support requirements identification and analysis for an IDS has been presented with supporting examples and illustrative uses. Division of fault trees for intrusions into seven stages was examined, and sample fault trees for each of the intrusion stages were described. Using these staged subtrees, two intrusions were examined and software requirements for detection of the attacks were derived from examination of the trees and associated minimum cut sets. An example use of SFTA for guiding countermeasures' requirements analysis was also described.

SFTA's enable structured analysis of intrusions, supporting both requirements evolution as new intrusions are added and prioritized, incremental development of a distributed, agent-based IDS.

We have begun to formalize the use of the developed software fault trees to drive the development of an intrusion detection design. Software fault tree models of intrusions provide an indirect requirements description for the design of the IDS. The resulting design is modeled on Colored Petri Nets and implementable in mobile agents. SFTA models of intrusions may also assist the verification process by providing testcase scenarios (paths of attack) that the IDS is required to detect.

References

[1] CERT Coordination Center. FTP bounce. Online, Dec. 1997. `http://www.cert.org/advisories/CA-97.27.FTP_bounce.html`.

[2] CERT Coordination Center. Two input validation problems in FTPD. Online, July 2000. `http://www.cert.org/advisories/CA-2000-13.html`.

[3] De Lemos, R., Saeed, A., and Anderson, T. Analyzing safety requirements for process-control systems. *IEEE Software 12*, 3 (1995), 42–53.

[4] Del Gobbo, D., Cukic, B., Napolitano, M. R., and Easterbrook, S. Fault detectability analysis for requirements validation of fault tolerant systems. In *4th IEEE International High-Assurance Systems Engineering Symposium* (1999), IEEE Computer Society, pp. 231–238.

[5] Dittrich, D. The DoS Project's "trinoo" distributed denial of service attack tool. Online, Oct. 1999. `http://staff.washington.edu/dittrich/misc/trinoo.analysis.txt`.

[6] Dittrich, D. The "stacheldraht" distributed denial of service attack tool. Online, Dec. 1999. `http://staff.washington.edu/dittrich/misc/stacheldraht.analysis.txt`.

[7] Dittrich, D. The "tribe flood network" distributed denial of service attack tool. Online, Oct. 1999. `http://staff.washington.edu/dittrich/misc/tfn.analysis.txt`.

[8] Dittrich, D., Weaver, G., Dietrich, S., and Long, N. The "mstream" distributed denial of service attack tool. Online, May 2000. `http://staff.washington.edu/dittrich/misc/mstream.analysis.txt`.

[9] Hansen, K. M., Ravn, A. P., and Stavridou, V. From safety analysis to software requirements. *IEEE Transactions on Software Engineering 24*, 7 (July 1998), 573–584.

[10] Helmer, G. *Mobile agent system for intrusion detection.* PhD thesis, Iowa State University, Ames, IA, USA, Dec. 2000.

[11] Helmer, G., Wong, J. S. K., Honavar, V., and Miller, L. Intelligent agents for intrusion detection. In *Proceedings, IEEE Information Technology Conference* (Syracuse, NY, USA, Sept. 1998), pp. 121–124.

[12] Jansen, W., Mell, P., Karygiannis, T., and Marks, D. Mobile agents in intrusion detection and response. In *Proceedings of the 12th Annual Canadian Information Technology Security Symposium* (Ottawa, Canada, June 2000).

[13] Kumar, S. *Classification and Detection of Computer Intrusions.* PhD thesis, Purdue University, West Lafayette, IN, USA, Aug. 1995.

[14] Kumar, S., and Spafford, E. H. A pattern matching model for misuse intrusion detection. In *Proceedings of the 17th National Computer Security Conference* (Baltimore, MD, USA, Oct. 1994), pp. 11–21.

[15] Leveson, N. G. *Safeware: System Safety and Computers.* Addison-Wesley, Reading, MA, USA, 1995.

[16] Lin, J.-L., Wang, X. S., and Jajodia, S. Abstraction-based misuse detection: High-level specifications and adaptable strategies. In *Proceedings, IEEE Computer Security Foundations Workshop* (Rockport, MA, USA, June 1998), pp. 190–201.

[17] Lutz, R., and Woodhouse, R. M. Requirements analysis using forward and backward search. *Annals of Software Engineering 3* (1997), 459–475.

[18] Manian, R., Dugan, J. B., Coppit, D., and Sullivan, K. J. Combining various solution techniques for dynamic fault tree analysis of computer systems. In *3rd IEEE International High-Assurance Systems Engineering Symposium* (1998), IEEE Computer Society, pp. 21–28.

[19] PUKETZA, N. J., ZHANG, K., CHUNG, M., MUKHERJEE, B., AND OLSSON, R. A. A methodology for testing intrusion detection systems. *IEEE Transactions on Software Engineering 22*, 10 (Oct. 1996), 719–729.

[20] RAHEJA, D. G. *Assurance Technologies: Principles and Practices.* McGraw-Hill Engineering and Technology Management Series. McGraw-Hill, New York, 1991.

[21] RUIU, D. Cautionary tales: Stealth coordinated attack howto. *Digital Mogul 2*, 7 (July 1999).

[22] STANIFORD-CHEN, S., CHEUNG, S., CRAWFORD, R., DILGER, M., FRANK, J., HOAGLAND, J., LEVITT, K., WEE, C., YIP, R., AND ZERKLE, D. GrIDS-a graph based intrusion detection system for large networks. In *19th National Information Systems Security Conference Proceedings* (Oct. 1996), pp. 361–370.

[23] VAN DIJK, P. How we defaced www.apache.org. Email Message, May 2000. `http://www.securityfocus.com/archive/1/58478`.

Appendix

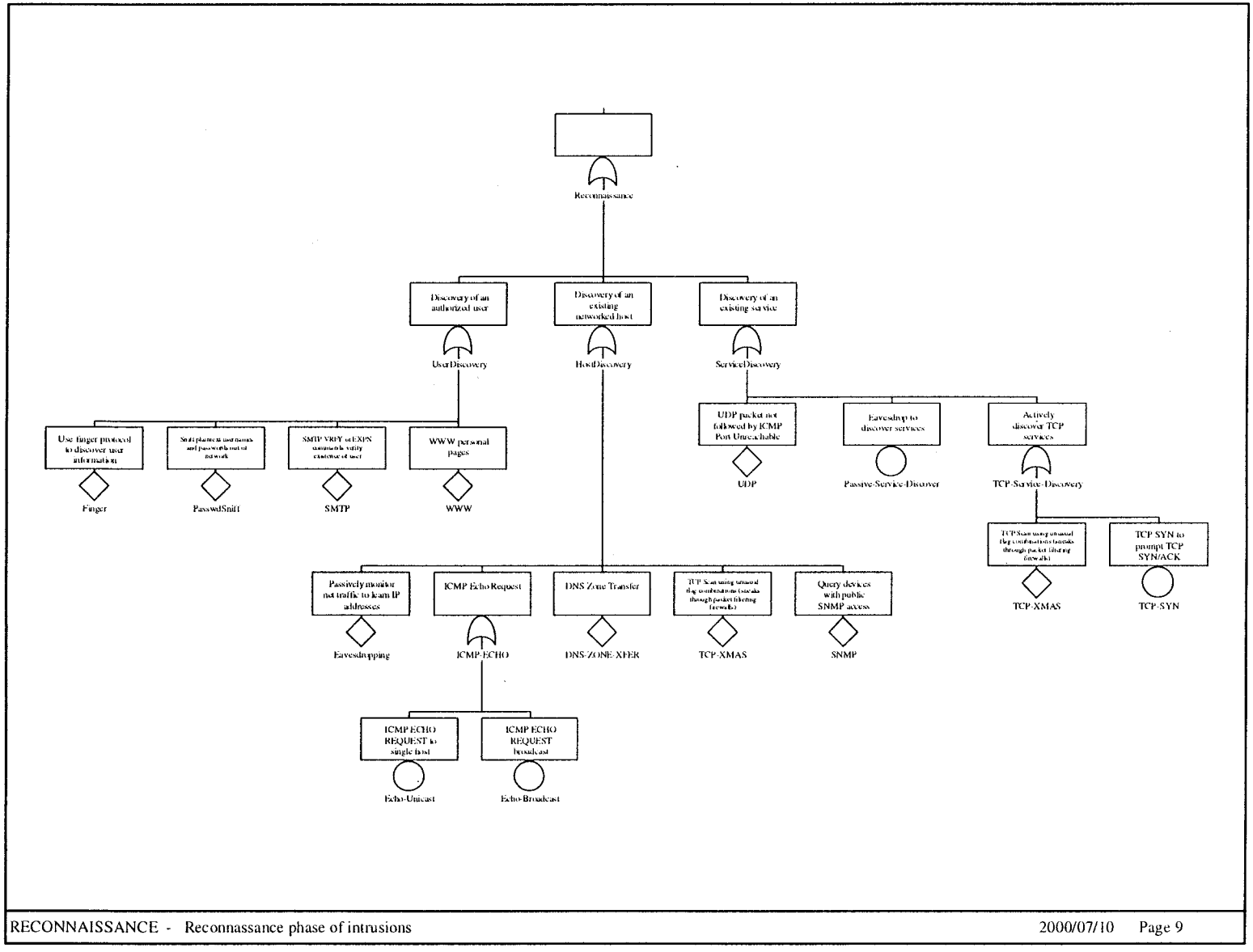


Figure 1: Reconnaissance fault tree

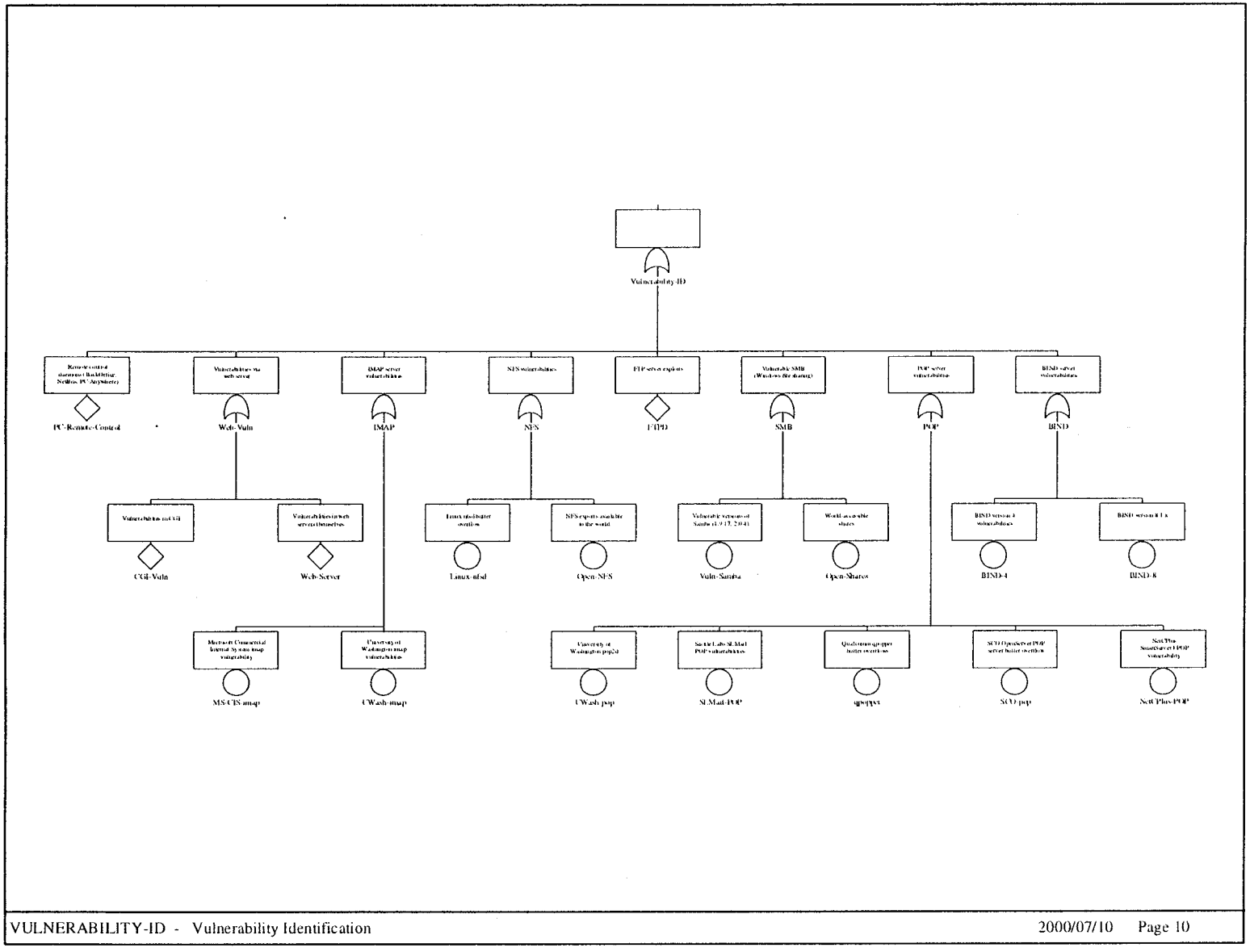


Figure 2: Vulnerability identification fault tree

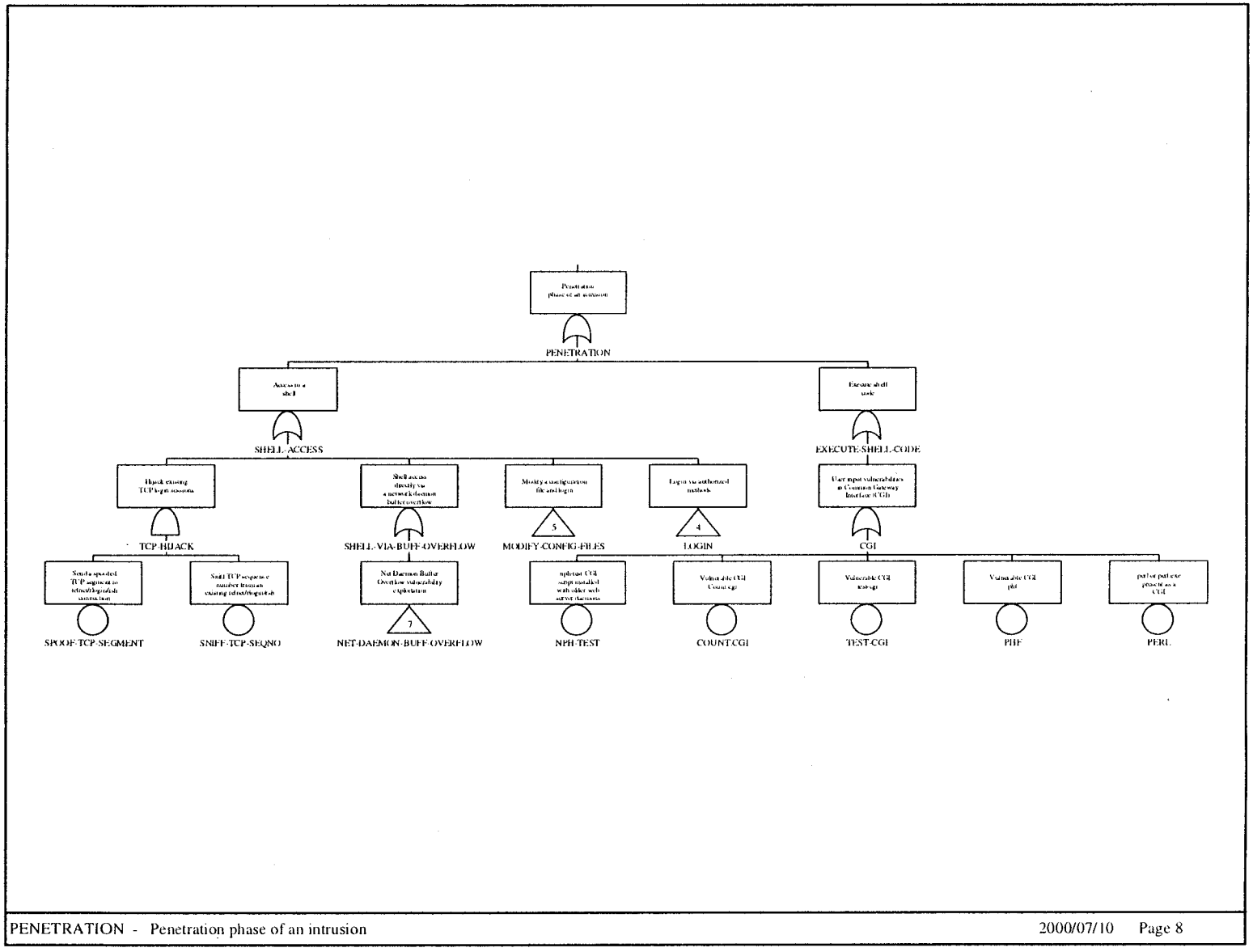


Figure 3: Penetration fault tree

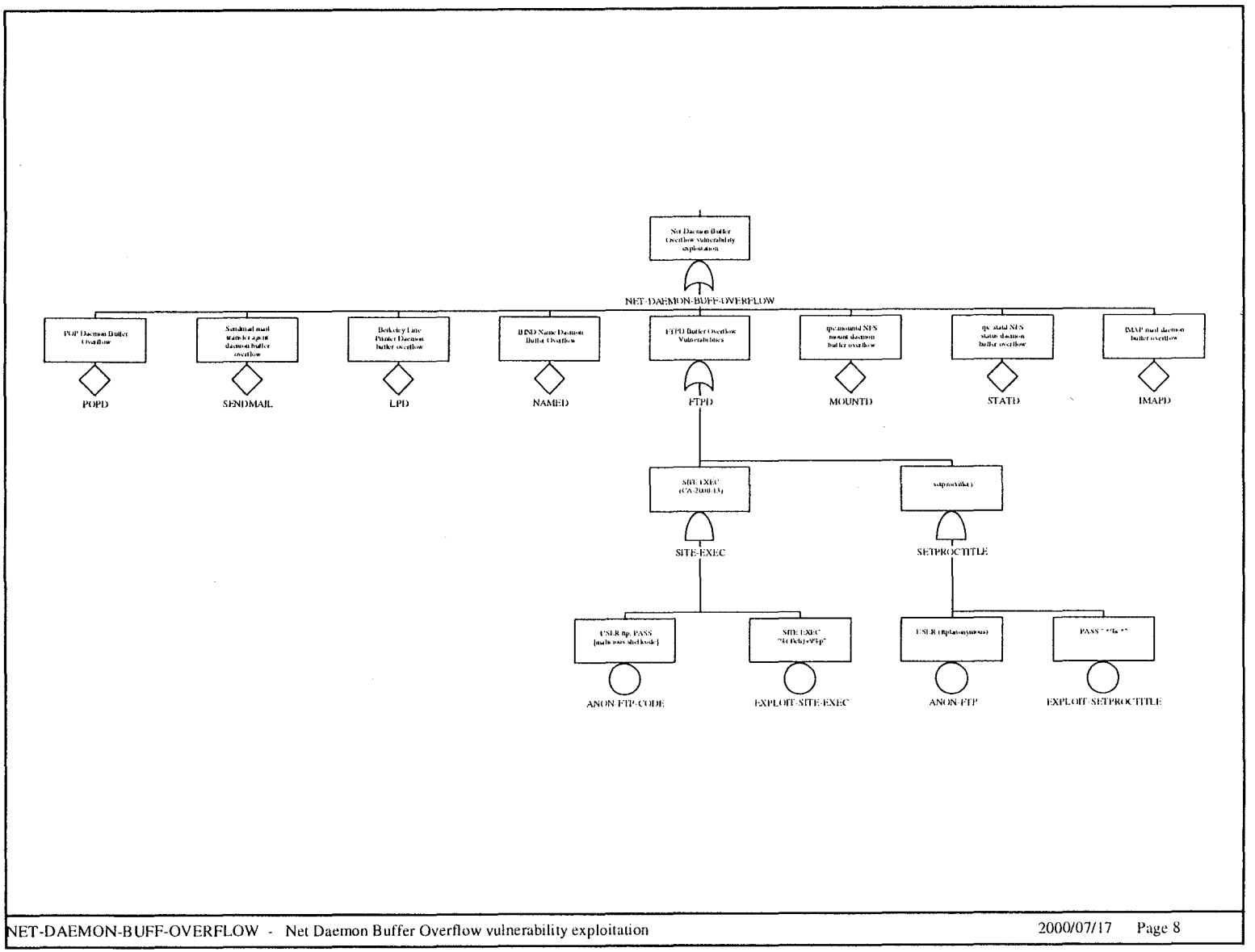


Figure 4: Penetration fault tree: Using buffer overflows in network daemons

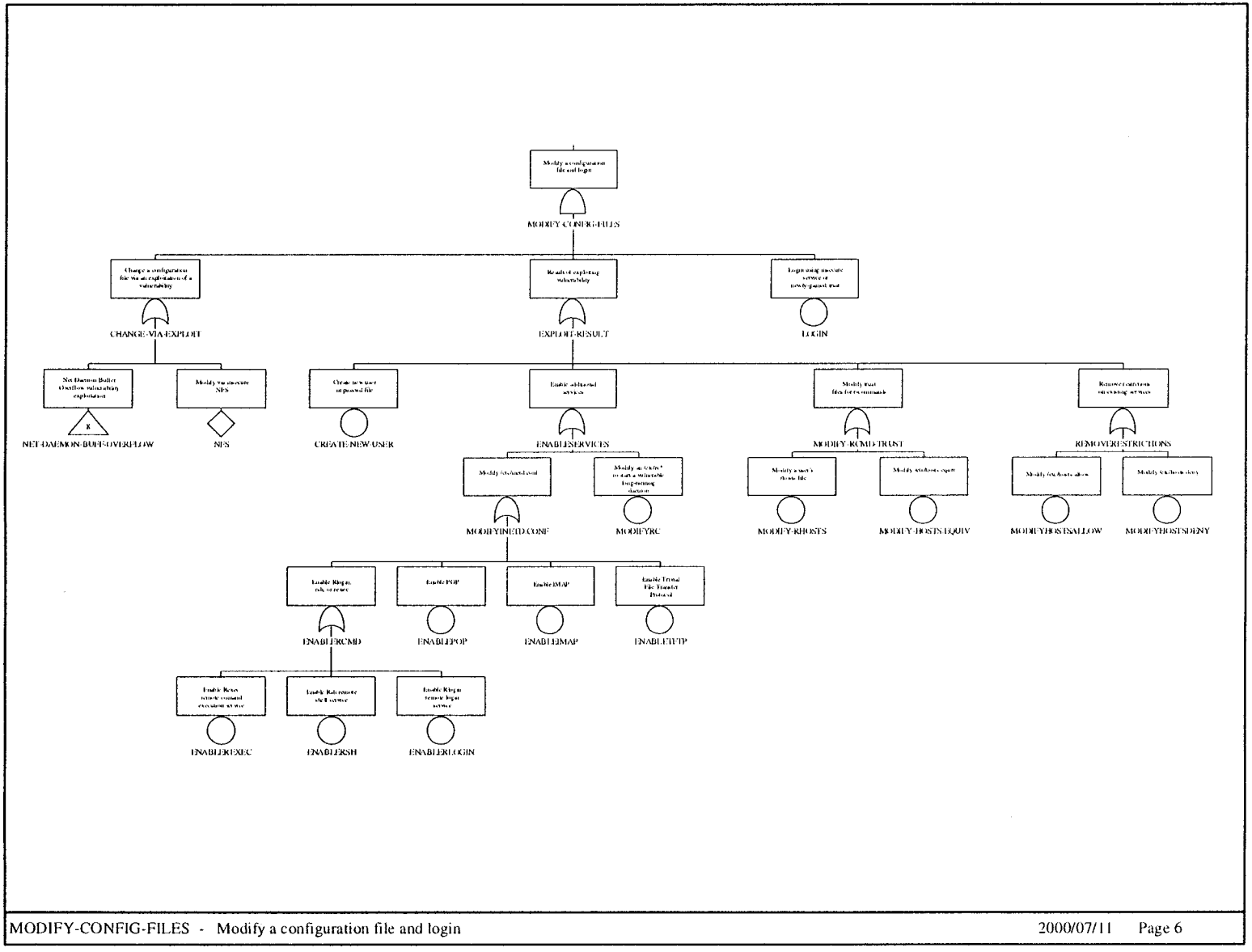


Figure 5: Penetration fault tree: Gaining access by modifying configuration files

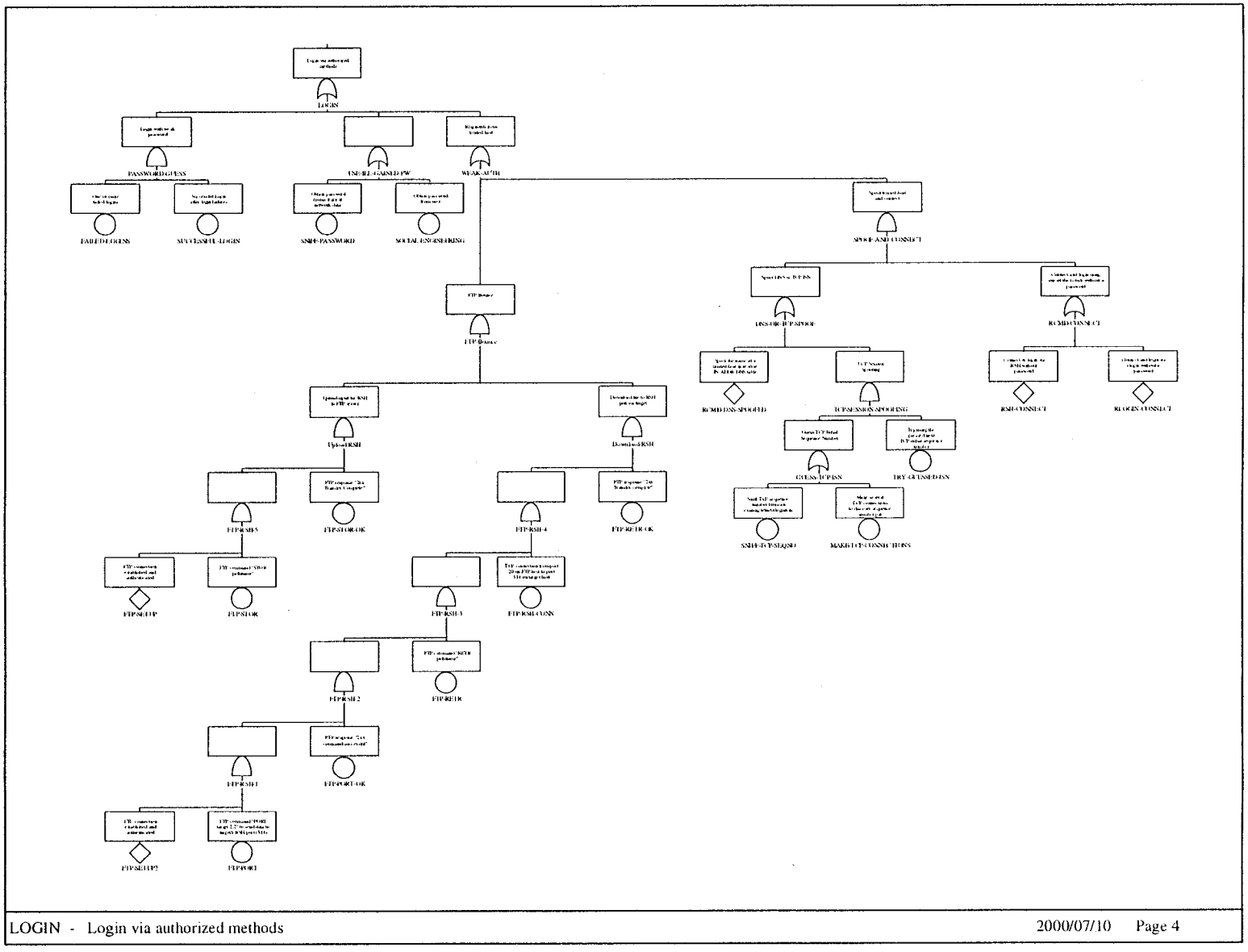


Figure 6: Penetration fault tree: Gaining access through abuse of authentication methods

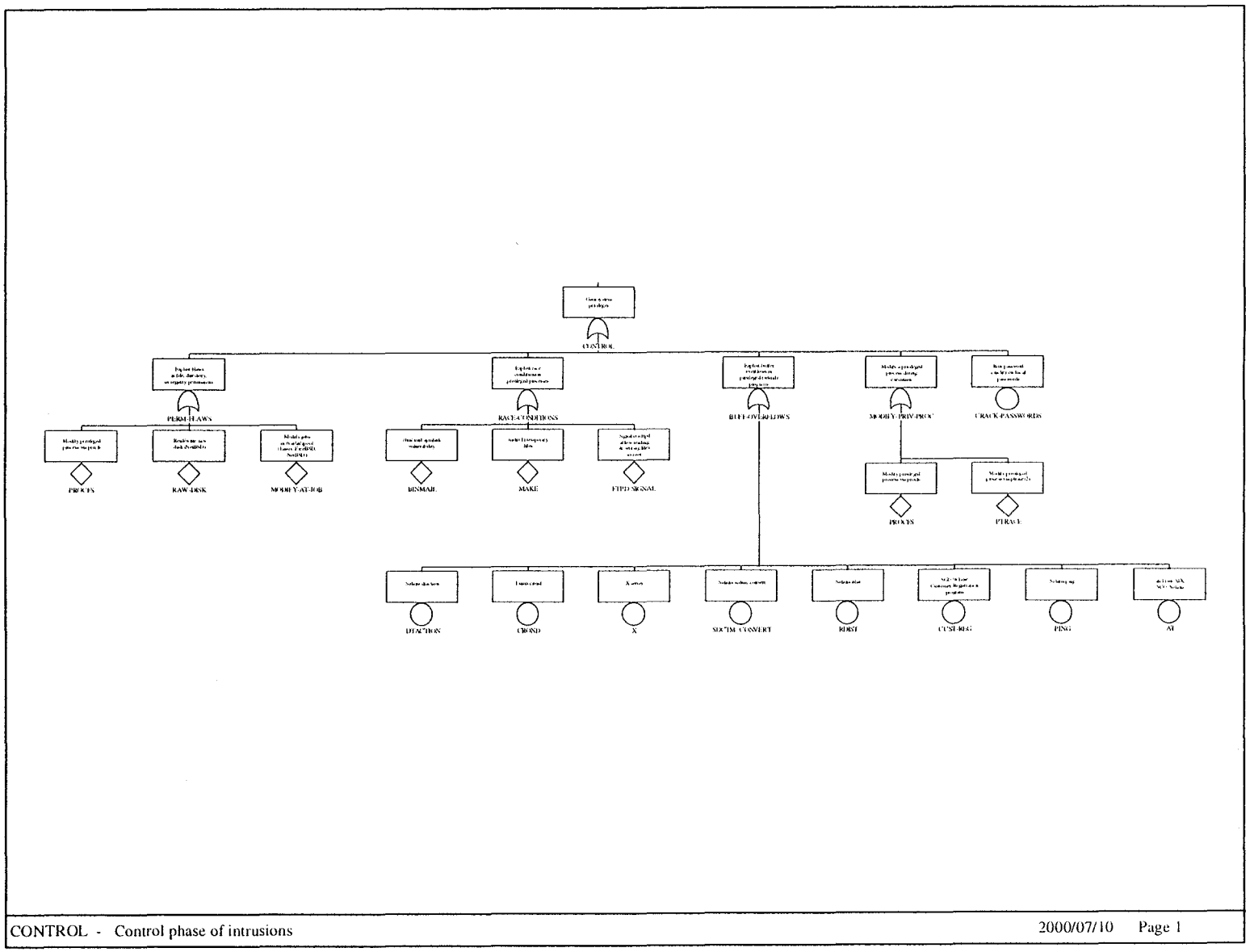
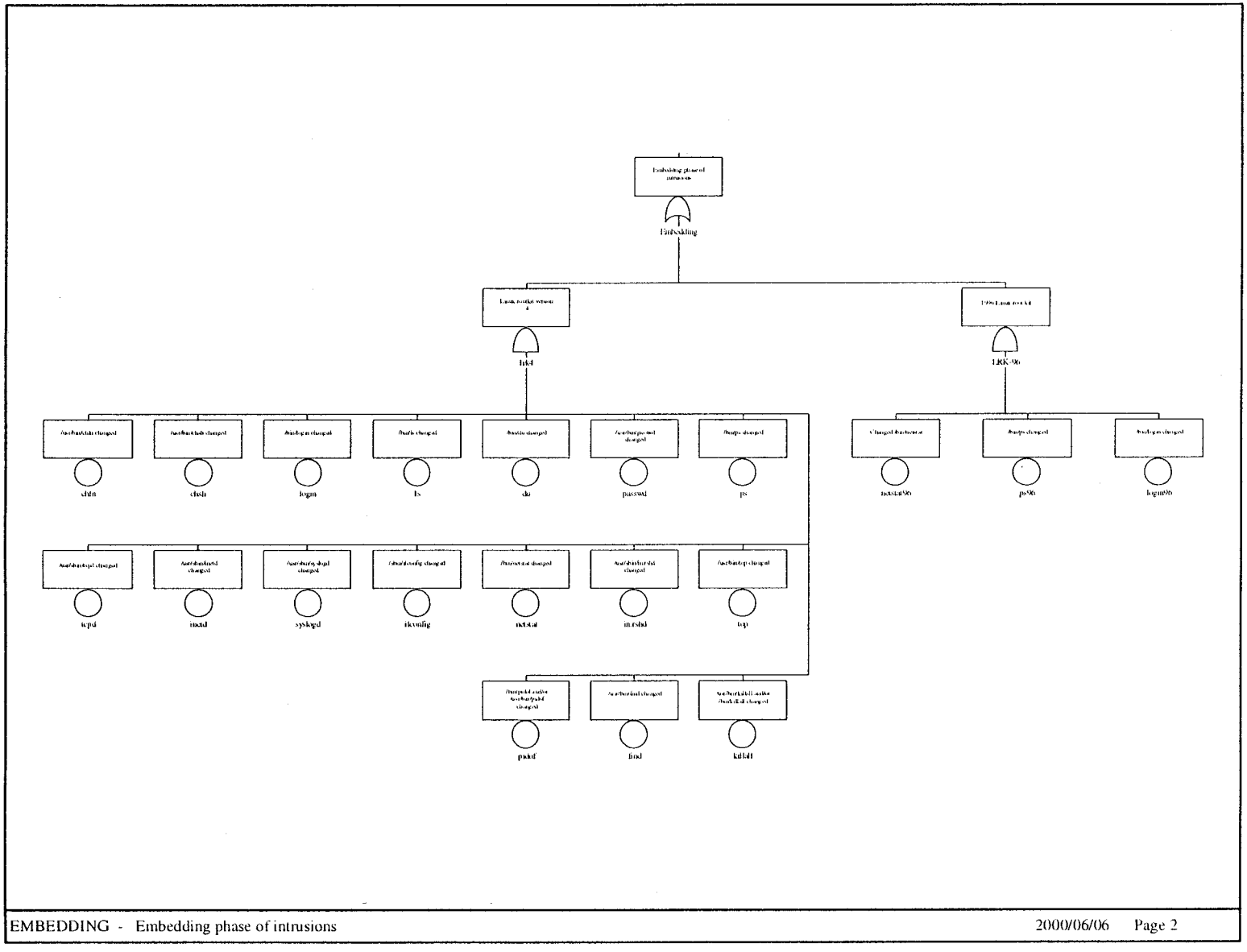


Figure 7: Control fault tree



EMBEDDING - Embedding phase of intrusions 2000/06/06 Page 2

Figure 8: Embedding fault tree

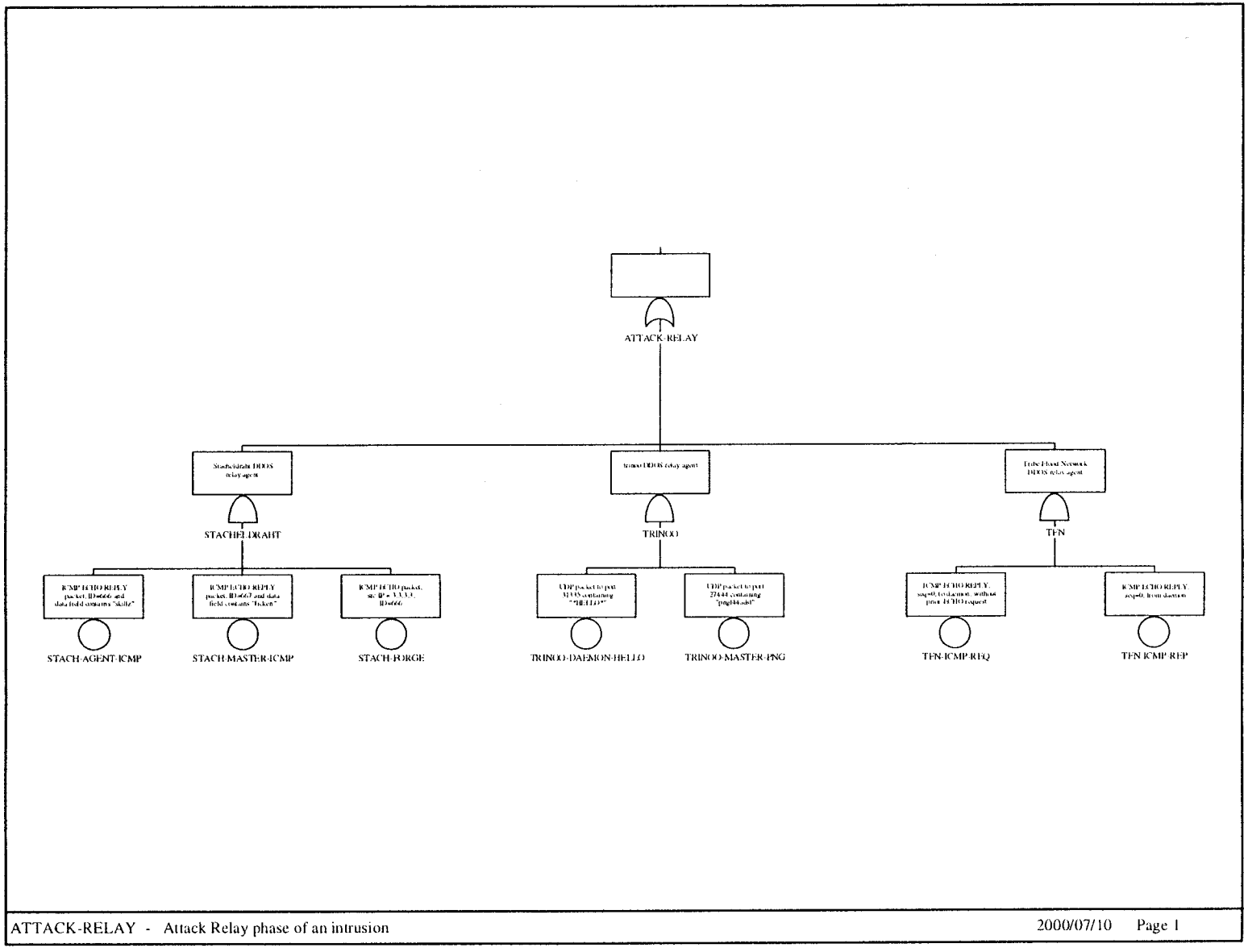


Figure 9: Attack relay fault tree