Tracing the Root of "Rootable" Processes

Amit Purohit, Vishnu Navda and Tzi-cker Chiueh
Stony Brook University

Abstract

In most existing systems, the authorization check for system resource access is based on the user ID of the running processes. Such systems are vulnerable to password stealing/cracking attacks. Considering that remote attackers usually do not have physical access to local machines, we propose a security architecture called NPTrace (Network-Wide Process Tracing), which requires a user to know the root password and to prove that he is within some physical proximity in order to exercise the root privilege. More specifically, NPTrace attaches a `Privilege-Level` *attribute to every process, and propagates this attribute across machines on demand. The Privilege-Level attribute of a process is set to Rootable if the system can trace back its origin to a process started by a user that has physically logged on from a specific set of hosts on the network. Only a root process with this Privilege-Level attribute set to Rootable, is allowed to perform privileged operations. The NPTrace architecture essentially exploits physical security to strengthen password-based security. This paper describes the design and implementation of the NPTrace prototype, which features a distributed mechanism to identify the entry point of a user into a network. The prototype is implemented under Linux and has been tested under many attack scenarios. The system shows correct behavior in these tests with negligible performance overhead.*

1. Introduction

Stealing passwords through automated cracking or social engineering poses a dangerous threat because once a password is stolen, most existing systems cannot distinguish an attacker behind a stolen password from its legitimate owner. In many cases, an attacker breaks into a machine and changes the password file so that she can later access the victim machine with the privilege of a forged account. The effect of this process is the same as stealing a password. If an attacker can log into a machine as the root, current systems can do very little to protect themselves, because their user authentication check is mainly based on password. One way to solve this problem is by adding another level of security on the top of the existing password security. Use of smart cards is one of the alternatives. But smart card suffers from the same drawbacks as that of password unless deployed accurately.

To protect a computer system from attackers that somehow possess the root password, we ask the following question: Is there any difference between a legitimate root user and a remote attacker that is disguised as a root user? One key difference is that a remote attacker, in most cases, is not within physical proximity of the victim machine, whereas a legitimate root user usually is. Therefore, if it is possible to tell whether a root process is initiated by a user that is physically close to a machine, one can then distinguish between authorized and unauthorized root users. One can generalize this idea by requiring that to become a root user on any machine in an intranet, one has to know the root password on that machine *and* to physically log into a set of machines that are well protected via existing physical security mechanisms. We call this set of machines *Physically Secure Subset(PSS)*. By *physically* logging into a machine, we mean the user needs to log into the machine through the console terminal.

Conceptually the above idea is no different from a standard computer security practice in which a system can reject any login attempts as root from a remote machine. However, this approach is too limiting as it also eliminates the possibility that legitimate root users can manage and maintain remote machines from a single host. To balance the convenience of remote management and more rigorous root access check, we propose the NPTrace (Network-Wide Process Tracing) security prototype that provides a more fine-grained root access control mechanism by taking into account the physical proximity of a user, in addition to the standard root password-based check. NPTrace attaches a Privilege-Level attribute to every process. It can take values: Rootable and Non-Rootable. Only processes that have their Privilege-Level attribute set to Rootable, are considered as a valid root user. In this prototype, when a user physically logs into a machine belonging to PSS, any process he initiates is a valid Rootable process. If in addition the user is a root user, all the processes he initiates possess root privilege. However, if a user logs in to a host outside the PSS

then the processes initiated by him are not Rootable, even if its owner is a root user, the process still does not possess root privilege. Note that PSS machines and the machines that trust them do not have to be physically close to one another. However, for security reasons they need to be able to communicate with each other directly.

A process's Privilege-Level attribute is automatically inherited by its descendant processes, like other process attributes. That is, the Privilege-Level attribute can be propagated *vertically* between parent and child processes. The Privilege-Level attribute can also be propagated *horizontally* between processes on different machines that are related through a remote login mechanism. For example, consider a process P on machine A which is a Rootable process. If it initiates a login to another machine B and starts another process Q on machine B, then process Q also inherits the Privilege-Level attribute of process P. Exactly how this is done in a secure way, is the main research focus of this project. The proposed prototype works on the assumption that users log into remote hosts via telnet or ssh, although it is fairly easy to extend it to support other services. We concentrate on these two services as they are the most widely used services for remote login. Even if there are other means for remote login they are generally disabled for security reasons. In summary, a process can perform root-privilege operations if and only if its user is a root user and it itself is Rootable.

Moreover, a process on a machine is Rootable if its user logs into the machine through the console, or he logs into one of the machines in PSS. When a Rootable process initiates a login session into another host using telnet or ssh, its Privilege-Level attribute is propagated to a process on the remote host only if the process is originated on a machine belonging to PSS.

We have successfully implemented the NPTrace prototype under Linux, and have conducted a series of tests of the prototype covering many different attack scenarios. The prototype is able to successfully distinguish between remote and local root users by correctly propagating the Privilege-Level attribute both vertically and horizontally. The rest of the paper is organized as follows. Section 2 describes the design and implementation of the NPTrace prototype. Section 3 provides performance results of the NPTrace prototype and discusses various attack scenarios. We discuss related work in Section 4. Section 5 concludes this paper and outlines future work.

2. Design and Implementation

We designed NPTrace with following goals in mind:

- **Security :**NPTrace is robust and detects the attacks under many complicated circumstances.
- **Performance :** NPTrace has minimal performance overhead and hence the overall system behavior is unaffected.
- **Simplicity :** The user interaction involved is minimal. NPTrace automatically determines the origin of a process.

2.1. Overview

Password check is the only form of authentication check in most current computer systems. This type of access control mechanism is insufficient because an attacker with stolen user name and password is treated no differently than a legitimate user. As a result, once an attacker acquires the root password of a victim machine, she can login into the system from anywhere in the world without being detected. To prevent such attacks, we need a way to distinguish between an authentic root login session and a login session by a remote attacker possessing root password. By definition, remote attackers do not have physical access to the victim machine. That is, in almost all cases an attacker comes into a victim machine from some remote host on the Internet. Therefore, if one can detect that a process is started by a remote user, stripping the process of root privilege will stop all remote attacks using stolen passwords. Based on this observation, we propose the NPTrace security prototype, which requires that a process can have root privilege if and only if (1) its origin is the on the local machine or on one of a special subset of machines that are well protected through physical security, and (2) its owner is a root user. NPTrace prohibits administrators from remotely logging into the system as root from arbitrary Internet hosts. Instead, they can become root on any machine in the network if and only if they first log into one of a subset of hosts via a console, and then telnet or SSH to other hosts from there. This special subset of hosts is called the `Physically Secure Subset` or `PSS`. A list of hosts that belong to PSS is specified in a configuration file that can be accessed/modified only by the root.

A process is Rootable, that is, it can have real root privilege in following cases.

- All the processes are Rootable on the host on which they originate.
- A process that has its origins on a different host is Rootable on the current host, if the host on which the process originated belongs to PSS.

The first criterion says that a user that has root password and is physically present in front of a machine is given full root privilege on that machine, even if the machine is not part of PSS. The second criterion states that a user is able to control all hosts in the intranet by first physically logging into one of the hosts that belong to PSS. This host is

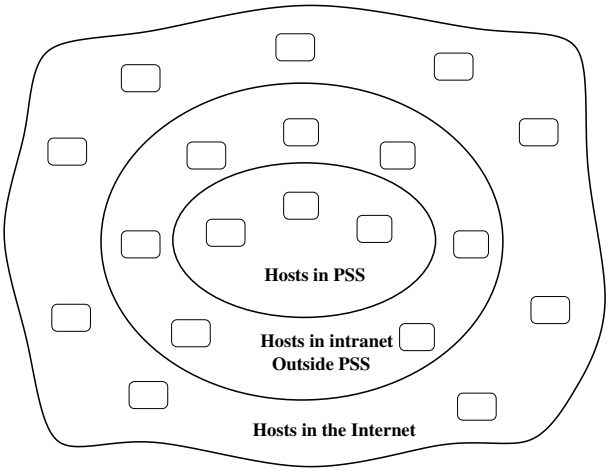

Figure 1. A Physically Secure Subset (PSS) of hosts is the subset that are protected by additional physical security mechanisms and thus less likely to be attacked physically.

called the originating host. When he later logs into some other hosts from the originating host, directly or indirectly, he still can have full root privilege on these hosts, because all processes he initiates on these hosts have their origin from a physical login process on a PSS host. The user of a physical login process on a PSS host is trustworthy because he must have passed physical security check, and is thus unlikely to be a remote attacker.

A root process started by a remote attacker via a stolen password does not qualify for either of the above two criteria, and thus will not have root privilege even when the attacker possesses the right password. By considering the physical proximity of a root user, NPTrace offers a more rigorous and flexible defense against stolen and guessed password attacks than existing systems.

To implement the NPTrace architecture, we need to keep track of the origin or "root" of each process. Following the parent-child relationship, we can first trace back to the oldest ancestor process on the same host. Then there are two possibilities: either this ancestor process belongs to a console session, or it is created due to a remote login. To trace the origin of a process across hosts, we need to define the notion of remote parent child relationship. We say a remote process is a parent of a local process if

- The two processes have a network connection between them.
- The behavior of one process is controlled by the other. For example, the remote process could send commands over the network and get them executed by the local process.

This general definition encompasses remote login sessions through SSH or telnet. That is, after a user logs into a remote machine using SSH, the SSH client process is a parent of the corresponding shell process running on the remote machine.

2.2. Privilege Attribute and Propagation

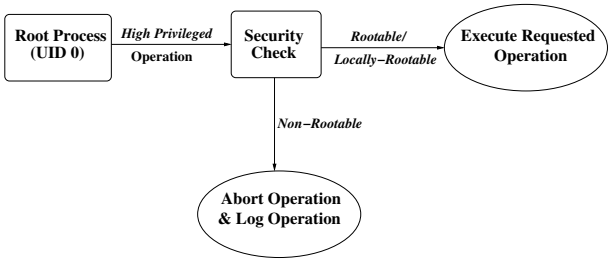

Figure 2. Authorization check is based on Privilege-Level attribute as well as password.

We associate an attribute, *Privilege-Level*, with each process to keep track of its origin information, which is derived using the extended parent-child definition, and serves as the basis for access control check. This attribute can take one of the following three possible values: `Rootable`, `Locally-Rootable` and `Non-Rootable`. If the origin of a process is a console session, and the machine belongs to PSS, its Privilege-Level attribute is set to Rootable. If the origin of a process is a console session, and the machine does not belong to PSS, its Privilege-Level attribute is set to Locally-Rootable. If the origin of a process is not a console session, and its remote parent is Rootable, its Privilege-Level attribute is set to Rootable. If the origin of a process is not a console session, and its remote parent is not Rootable, its Privilege-Level attribute is set to Non-Rootable.

Whenever a process makes a request for a privileged operation, two authorization checks take place, as shown in Figure 2. One is the existing system authorization check to determine whether the process is owned by a root user or not. The second is a new check based on value of the process's Privilege-Level attribute. If the value is Non-Rootable, it means that the process is being controlled by a remote host that does not belong to the PSS and therefore the request is denied. If the value is Rootable or Locally-Rootable, it means that the process is controlled by a user that physically logs into the local machine or a PSS machine, and therefore the request is allowed. All the set of operations that require root privilege will go through this two-step authorization process.

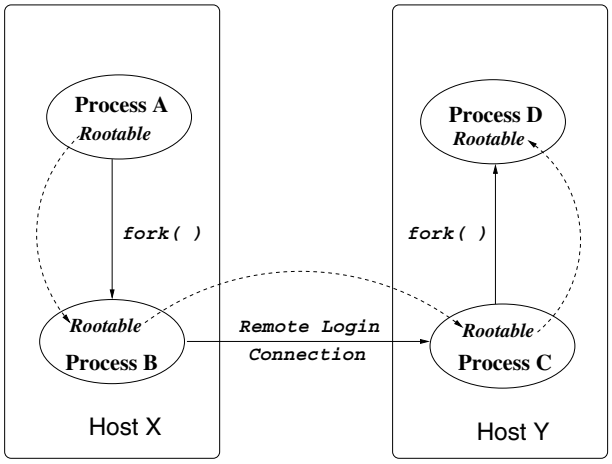

Figure 3. Vertical propagation of Privilege-Level attribute occurs between a parent and child process during fork. Horizontal propagation of Privilege-Level attribute occurs between processes on different machines that are involved in a remote login session.

Propagation of the Privilege-Level attribute value takes two forms. During *vertical propagation*, when a process forks, the child process inherits the Privilege-Level attribute from its parent. During *horizontal propagation*, when a process on one machine logs into another machine, the Privilege-Level attribute of the source process is propagated to the new process on the destination machine. Figure 3 explains the propagation of privilege level attribute. The Privilege-Level attribute for process A is set to Rootable. When process A forks to create process B its privilege is propagated to the child process. Hence, process B also becomes Rootable. This is the vertical propagation of the attribute. When process B established a remote connection to a process C on host Y, its Privilege-Level attribute is propagated to process C. This is the horizontal propagation of the attribute. The Privilege-Level attribute further propagates to Process D vertically.

2.3. Vertical Propagation

We intercept the fork system call to pass the Privilege-Level attribute value from the parent process to the child process. At system start-up, all child processes of the init process (PID 0) are set to Locally-Rootable. Consequently all daemon processes and their children are set to Locally-Rootable, and thus can perform privileged operations. This setting is necessary for daemons to work properly, as they require root privileges to perform their tasks. Also, processes that belong to a console session, and their chil-

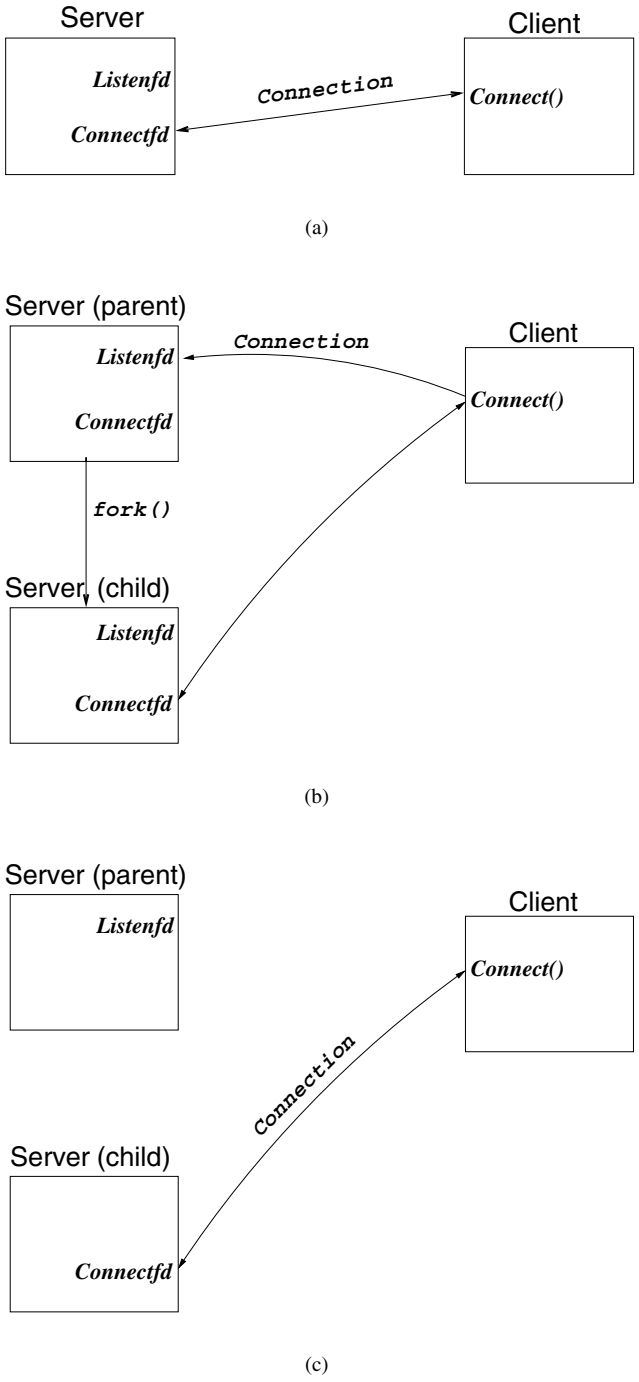

Figure 4. After a server accepts a connection request from a client, it first establishes a connection with the client (a), then forks a new process that inherits that connection (b), and finally closes the connection (c) so that the new process is the only entity that uses the connection.

dren, are also set to Locally-Rootable. Therefore, processes started by a user that physically logs into a machine are Locally-Rootable.

2.4. Horizontal Propagation

To explain how horizontal propagation works, we first show the client-server communication dynamics of a remote login session. Initially a server creates a socket bound to a well known port and sets the socket as a listening socket, as shown in Figure 4(a). It makes a blocking `accept` system call. When a client send connect request on this port, the server comes out of the `accept` system call with a new connection socket being created. This socket has all the details about the connection (Src IP, Src Port, Dest IP, Dest Port). It then forks a new process to handle this connection, as shown in Figure 4(b). After that, the newly created process takes care of the new connection and the parent returns to the blocking `accept` system call, as shown in Figure 4(c).

Outline of a typical concurrent server [12].

```
pid_t pid;
int listenfd, connectfd;

listenfd = Socket( ... );

/* fill the sockaddr_in() with server's
   well-known port  */
Bind(listenfd, ... );
Listen(listenfd, LISTENQ);
for(;;) {

 /* blocking call */
 connectfd = Accept(listenfd, ... );

 if ( (pid = Fork()) == 0) {
  Close(connectfd);  /* child closes
                      listening socket */
  doit(connectfd);   /* process the request */
  Close(connectfd);  /* done with the client */
  exit(0);        /* child terminates    */
 }

 /* parent closes connected socket      */
 Close(connectfd);
}
```

As a policy decision we need to know the server program that is used to service remote login requests. In the current prototype, we assume it is either telnetd or sshd. To know the process ID of these two daemon processes, we modified the startup script for these daemons so that they register their PID with the NPTrace module inside the kernel at start-up time. When telnetd/sshd accepts a connection and does a fork, the system call monitoring module intercepts this call and determines the PID of the process which initiates the call. If it is a telnetd or sshd, the system determines the Privilege-Level attribute value of the process on the remote machine that initiated the connection, and based on this, sets the Privilege-Level attribute of the child process that telnetd or sshd forks. Table 1 specifies how the Privilege-Level attribute is propagated from a remote process to a local process.

Remote Host belongs to PSS	Privilege level of remote process	Privilege level of local process
YES	Rootable	Rootable
YES	Locally-Rootable	Rootable
YES	Non-Rootable	Non-Rootable
NO	Rootable	Rootable
NO	Locally-Rootable	Non-Rootable
NO	Non-Rootable	Non-Rootable

Table 1. The rules in this table are used in Privilege-Level attribute propagation across a remote login operation.

By looking through the process structure, NPTrace gets the open socket descriptor for a socket connection, and then the IP address and the port number of the remote host involved in the socket connection. Then a query is made to the NPTrace daemon running on the remote machine to find out the value of Privilege-Level attribute of the remote process that invokes the current connection (Src IP, Src Port, Dest IP, Dest Port). Given a port number, the NPTrace daemon scans all processes to identify the process that is currently bound to the given port. For each process, it finds the open socket descriptors and checks if there is a match. The matching process is the originator of this remote login session. From the process structure, NPTrace retrieves and passes the Privilege-Level attribute information to the NPTrace daemon at the requester end. The communication between user-level NPTrace daemons is protected using Open-SSL to prevent interception and man-in-the-middle attacks.

2.5. Example

To illustrate the operation of the NPTrace prototype, we explain a complete telnet session example. The communication among different NPTrace entities is shown in Figure 5.

1. Process P1 on host X initiates a telnet connection to Host Y.
2. Upon successful authentication, telnetd forks a shell process, P2, on Host Y.
3. Fork system call is intercepted and based on the parent process id, the system call monitor detects that the

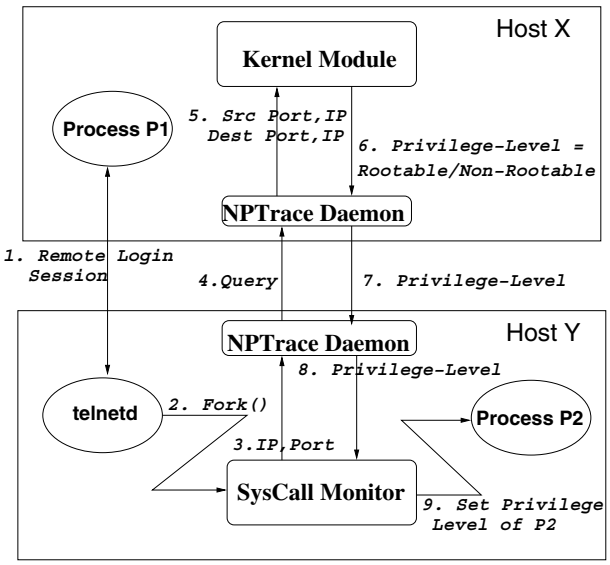

Figure 5. Communication occurring between NPTrace daemons (user and kernel) on end hosts, during Horizontal propagation of Privilege-Level attribute

process is telnetd and examines the open socket descriptor structure held by the parent process to determine the IP address and the port number of the remote host. The kernel module forwards this information to the user-level NPTrace daemon to ask the remote host to determine the Privilege-Level of the process which initiated the connection.

4. The NPTrace daemon on host Y queries the corresponding daemon on host X with the Src IP, Src Port, Dest IP and Dest Port of the open socket connection.

5. The NPTrace daemon on host X, upon receiving the request, relays the query to the kernel module, which compares the socket description with each open socket descriptor on host X, until it locates the process that initiated the socket connection. The kernel module determines the value of the process's Privilege-Level attribute and sends it to the user level daemon.

6. The NPTrace daemon on host X relays the Privilege-Level attribute information back to the daemon on Host Y, which in turn relays it into the kernel module. The IP address where the process originated is also attached along with the Privilege-Level attribute.

7. In case the process is returned as Rootable, Host Y communicates to the originating host to confirm the request for new connection. After receiving the confirmation, the new process created has its Privilege-Level attribute set to Rootable. The Privilege-Level attribute of P2 is propagated to the process P1.

2.6. Enhancement

When an attacker compromises a host and gains root privilege, she can tamper the user-level NPTrace daemon and the kernel. Assume the compromised machine is H1, and the attacker attempts to log into a remote machine, say H2, as root. In this case, when H2 asks H1 whether the process that initiates the remote login session is Rootable, the tampered NPTrace daemon could "lie" and thus enable the attacker to be inside H2 via a Rootable process. That is, the attacker can artificially mark all local processes as Rootable, even though the compromised machine is not a PSS machine, and the attacker is not even Locally-Rootable on the compromised machine. More generally, the key research challenge here is, how the NPTrace architecture can still work when some of the non-PSS machines are compromised.

The way we solve this problem is by generating a unique key for each login process of a console session on a PSS machine, and propagating this key together with the Privilege-Level attribute both vertically and horizontally. Each key should include the IP address of the host on which it is created. When a Privilege-Level attribute is passed horizontally, the receiver machine first checks if the host that created the propagated key belongs to PSS. If the host does belong to PSS then it checks the validity of the Privilege-Level attribute by consulting with it. Only hosts within the PSS are authorized to generate key. They are protected by physical security and thus are not likely to be compromised by a remote attacker. When a key-generating host receives a request for validation, it searches the list of active keys it generates. If a match is found, it sends a confirmation request to the login process with which the key is generated. Only when the user confirms it, will the system respond positively to the machine that requires key validation. Asking for confirmation is appropriate because the user supposedly has already logged into the key-generating host, AND is currently present in front of the host (otherwise there should not be any login attempts).

Suppose a Rootable process (P1) on a PSS machine (H0) logs into a non-PSS machine (H1), and K1 is the associated key. When an attacker later compromises H1, she can "steal" K1, and use it to log into another machine, H2. But this attack will not succeed, because the legitimate user behind P1 will not confirm the request for key validation since he does not intend to log into H2. However, assume that the user behind P1 and the attacker on H1 both attempt to log into H2 at the same time. Moreover, the login attempt from the user behind P1 is suppressed. In this case, H2 sends a key validation request to H0, which eventually reaches the

user behind P1. Since the user indeed attempts to log into H2, he confirms it, not knowing that he actually confirms the login attempt by the attacker. To defeat this attack, the password check of telnet or SSH must precede the key validation check. By imposing this order, a user will be prompted for password before confirmation. Therefore, any key confirmation request that is not preceded by password prompting is a sign of hijacking, and thus should not be approved.

3. Evaluation

To evaluate the usefulness and validity of the NPTrace prototype, we conducted a series of tests corresponding to many different attack scenarios. In this section, we discuss the results of these tests.

3.1. Experimental Setup

Our experimental testbed involved 3 P-III 900MHz machines with 256MB of RAM. The machines were running different flavors of Linux kernel. We installed the NPTrace daemon on these machines. We also applied the NPTrace kernel patch to the vanilla Linux kernel. All the user activities, periodic jobs, and unnecessary services were disabled. We ran the experiments many times to confirm the stability and reliability of the system. All the network services, that provide remote login except sshd and telnetd, were disabled. First, we explain the behavior of NPTrace under different test scenarios and then, we demonstrate various overheads added by it and their impact on the overall system behavior.

3.2. Tests

Remote Login Cases : This test demonstrates the correctness of the NPTrace prototype. In this test, we used three machines M1, M2, and M3. All the machines were NPTrace enabled. The machines M1 and M2 belong to the PSS, but M3 does not. We simulated an attack from a user on machine M3 to machine M1 through M2. The user, who physically logs on to M3 uses SSH to establish a connection to M2 and then to M1. As the attacker has her origin on M3, the processes on M1 that belong to her are Non-Rootable and thus are not able to inflict any harm on M1.

We also tested the system with a slight variation of the above test case. In the second scenario, we disabled NPTrace on the attacker (M3) machine. This corresponds to the scenario in which an attacker logs into a victim host from a non-NPTrace machine. In this case, while establishing the login session from M3 to M2 the NPTrace daemon on M2 tries to retrieve information from the remote host M3. M3 being NPTrace disabled could not respond and hence the attribute was set to Non-Rootable for the newly started process on M2.

In the next test, the user physically logs into M2 and then establishes a login connection to M3 and then to M1. All the three machines are NPTrace enabled. As before M1 and M2 belong to the PSS and M3 does not. In this case, processes on M1 belonging to the user from M2 are allowed to be Rootable because their owner originates from a machine belonging to PSS.

Local Login Cases : This test shows the behavior of Locally-Rootable processes. In this test we use the same three machines. M1 and M2 belong to PSS but M3 does not. When the user physically logged on to M3, its Privilege-Level attribute was Locally-Rootable in spite of the fact that the machine M3 does not belong to the PSS. So, the Privilege-Level attribute can be propagated vertically within M3, but not horizontally. That is, when a Locally-Rootable process on M3 logs into other machines it cannot pass on its Privilege-Level attribute as Rootable. This test shows the selective propagation of the Privilege-Level attribute for Locally-Rootable processes.

3.3. Performance Overhead

NPTrace introduces additional overhead during the setup of a remote login session. We performed a micro-benchmark to measure this overhead. We calculated the extra time required due to the NPTrace prototype to setup a remote login session using ssh. We repeated this test 10 times to obtain statistically reproducible results. This overhead has to be paid at the start of a remote login session for both Rootable and Non-Rootable processes. If a user physically logs into host C, and later attempts to log into host B from Host A, then the additional overhead that NPTrace introduces includes are:

1. **Overhead due to communication between the user daemons on end hosts A and B:** Two messages are exchanged where host B requests for the Privilege-Level attribute and the process-origin information from host A.
2. **Time spent in the interaction between user daemon and kernel on both hosts A and B:** To find out the remote parent of the process, the user daemon on host B asks the kernel for the remote port number, and the user daemon on host A retrieves the Privilege-Level attribute and process-origin information from the kernel based on the port number.
3. **Overhead associated with the key validation process:** Host B communicates with the key generation machine, host C, to validate the authenticity of the reply from host A.

A remote login session using ssh takes an additional 12.5 msec when using NPTrace. This additional time includes overheads for all the above mentioned checks. The total number of additional messages exchanged per session over the network is 4, which is quite modest. Compared with the network bandwidth spent after a successful login, this overhead is quite small and should not pose any burden on a modern LAN.

We believe that for other remote login services (e.g. telnet) the overhead would be similar, as the same mechanism is followed to retrieve and verify the Privilege-Level attribute in both cases. Both of them share similar mechanism to set up a session, and thus should involve the same number of message passing among NPTrace entities.

The overhead of 12.5 msec is the only overhead that NPTrace introduces, and impacts only the session startup time. Hence, only those processes that try to establish a connection to remote hosts are affected. All the other processes are unaffected. During vertical propagation of the Privilege-Level attribute, the fields in the task structure of the parent are also copied in the task structure of the child. This requires some CPU time. But this overhead is equivalent to 12 bytes of memory to memory copy, and is thus very small.

3.4. Attack Analysis

In this subsection, we discuss the behavior of NPTrace under various attack scenarios.

Eavesdropping An adversary could try to figure out the NPTrace user-level daemon communication protocol between two hosts by snooping the channel. He could impersonate a host responding with a valid Rootable token to gain Rootable privilege on a host in PSS. To prevent such attacks, the NPTrace daemons communicate through a secure OpenSSL channel in the current implementation. There is one extra check from the requesting host to the originating host to confirm that no malicious user is faking the communication.

Daemon Tampering Attack An attacker could reverse-engineer the binary image of the NPTrace daemon and modify it into a Trojan binary that could impersonate a legitimate NPTrace to give out faked Rootable tokens. In the extreme case, if the attacker manages to replace the user daemon in such a way that, it does not break the authentication but still manages to ”lie” to the requesting hosts, NPtrace will still be able to detect such “lies” when the requesting daemon communicates to the originating host to confirm the reply, as explained in section 2.6.

Kernel Tampering Attack We will first explain the kernel tampering attack. Suppose the attacker has root access to a machine M1 which is outside PSS and he is trying to gain root access on a host machine M2 that belongs to PSS. Since attacker has root privilege on host M1, she can modify the kernel any way he wants. In this case, the attacker could modify the kernel so that all the processes are Rootable. As a result, when she remotely logs into host M2, the process that results from the login will also become Rootable. NPTrace solves this problem using scheme explained in section 2.6. NPtrace generates a key for each login process from a console on a PSS machine, and propagates this key together with the horizontal and vertical propagation of the Privilege-Level attribute. Each key includes the IP address of the host on which it originated. When a Privilege-Level attribute is passed horizontally, the receiver machine checks the validity of the Privilege-Level attribute by consulting with the originating host of the associated propagated key directly. Because a key generating host is guaranteed to be in PSS, it is protected by physical security and thus is least likely to be compromised by a remote attacker.

Buffer Overflow Consider a scenario where a Rootable process is vulnerable to `buffer overflow` attack. After an attacker successfully compromises this process through a buffer overflow security hole, she effectively has control of a Rootable process, even though she is from a machine outside the PSS. The reason NPTrace cannot help in this case is because the attacker does not log into the victim machine through telnet or SSH.

Backdoors After a successful buffer overflow attack the attacker generally tries to create a backdoor for herself. This backdoor is typically a program that listens on a port for commands from remote attacker. After the successful launching of a backdoor program, the attacker can execute arbitrary commands through the backdoor. NPTrace does not solve this problem, because once a process becomes Rootable, its Privilege-Level attribute is blindly copied to the children without any further checks.

4. Related Work

The related work section is divided into three sections. In the first section, we distinguish our work from the approaches of network traceback. In the second section, we compare our approach to the approaches that use security extensions to the operating system. The third section explains secure authentication systems.

4.1. Network Traceback

The goal of network traceback research is to allow determination of the source of attack traffic, so that a particular host used by a human to initiate an attack can be identified. One of the techniques [2, 13] which is being widely explored is to collect the traceback information at the routers to allow traceback of DoS traffic. Other methods [3, 6] add

marking to the packets to probabilistically determine the source, given a sufficient packets. We have borrowed the concept of traceback to identify the host on which the process originated.

Intruders generally log into the target machine through a chain of multiple computer systems to hide their trace. A paper [10] discusses one approach to locate the origin by tracing through the chain. By comparing the packet logs of the intruder on one machine with all the other recorded logs, they found out the deviation. If the deviation is small then the two hosts belong to the same chain. This method is not efficient when the data is encrypted. A similar approach [14] makes use of timing of packets to find the chain. Hence, it could be used in case of encrypted data. Another method [7] compares the rate of sequence number increase in TCP streams as a matching mechanism, which works as long as the data is not compressed at different hops and does not see excessive network delay. These methods rely on the network information which could be tricked using the well known methods to defeat network intrusion detection systems. The problem with these approaches is that, they rely on techniques which cannot uniquely identify the originating host and most of these techniques are offline. NPTrace requires accurate and instantaneous identification of the originating host and hence the above methods are not useful in our approach.

One of the recent attempts [4] tries to solve the problem of network traceback by finding the process origin. Every process is associated with the information about its origin and thus the audit information is enhanced by logging the origin and destination of network sockets. This is the only approach that tries to solve the problem of network traceback by monitoring each process. We are using similar concept of finding the origin of a process but the intentions differ. We are trying to solve the security problems like password attacks and privilege escalation. They are using the log information to solve problem of network traceback to find a system causing DoS attack.

4.2. Fine-Grained Access Control

There has been some work [1, 5] that attempts to use existing system information to match active incoming and outgoing streams. But this work has been shown to be impractical to securely implement.

Extensions[11][8] to Unix Security Model exist to provide fine-grained access to privileged commands. Capability Bounding Set[11] is a kernel based access control scheme on Linux, which defines a set of rules that are assigned to processes, users, files that even a root user must follow. Once a capability is removed from the bounding set, it may not be used by any process on the system, not even a process owned by root. A user-level program can use this feature to restrict itself to only those privileges it really needs, and dropping all other capabilities. Once dropped, neither the user-level program nor binary it spawns will be allowed to perform privileged operations, regardless of whether the program is running as root or not. The drawback of this technique is that they allow only one of the two options to exist at a time, a capability could be either completely removed or completely retained. If a particular privilege is dropped, even a valid root process is restricted from using that privilege. These extensions restrict privileges of a root user. So, even a malicious root user can only access system resources exposed to him, and rest of the system is protected. These techniques curtail the damage but cannot eliminate it completely. In our approach, instead of restricting the privileges of a root user, we ensure that user is genuine and allow full access.

4.3. Secure Authentication Systems

In some secure environments RSA/DSA authentication protocol [9] of OpenSSH is used as an alternative to password based login schemes. This protocol makes use of a private and a public key pair to establish a secure connection to remote systems without the need to supply a password. When a host tries to log into a remote host, the remote host uses the public key to challenge the requesting host. Requesting host uses its private key to decrypt the challenge and replays the deciphered challenge. Thus the remote host is assured of the requesting host's authenticity. The private key is extremely crucial for this scheme to work and has to be safeguarded from theft, as it could lead to unauthorized entry into the system. The private key is stored in encrypted format in the home directory or is stored offline on some device such as floppy disk. This scheme makes it harder for the attacker to log into the system since, he has to know the unique private key for the user account he is trying to hack into. Security provided by smart cards also falls in this category. These security mechanisms try to make the login process extremely secure so that nobody can ever break this protocol. This is a very strong assumption and NPTrace does not rely only on the security of passwords, private keys or the two fold security provided by such mechanisms. Passwords could be stolen through automated cracking or social engineering. Secret Keys stored in home of the user could be accessed maliciously. Our approach is applicable even when attacker already knows a way to login to such a secure system, where the above strong security mechanisms are of no use.

5. Conclusion

In this paper, we present the design, implementation, and evaluation of a secure login mechanism that leverages phys-

ical security to effectively stop remote attackers from gaining root privilege even when they know root password. The proposed secure login prototype, called NPTrace, requires a minimal amount of modification to the kernel, which includes addition of an Privilege-Level attribute to the process structure and a module to perform system call monitoring. The prototype implementation is integrated seamlessly into the existing system and is completely transparent to the users. The NPTrace architecture enforces an invariant that every Rootable process can be traced back to a process that is either a local daemon process or a remote process that has its origin in a host that belongs to a physically secured host set called PSS. A fully operational NPTrace prototype shows that the proposed architecture can indeed prevent all remote logins with stolen root password from having root privilege, with a negligible performance overhead. In addition to tightening up security, the Privilege-Level attribute of a process can also be used in many other applications such as network traceback and better system management and administration.

The current NPTrace prototype heavily relies on correct identification of remote login service processes like sshd and telnetd, which is currently based on process ID. Our present focus is on developing a robust mechanism for detection of such processes. A more general approach has to be developed to establish remote parent-child relationship, not just for telnet, SSH or similar services. We need to develop a mechanism that could identify that a process is performing some operations based on what it receives over a socket connection from another remote process. If this association can be reliably established, one could then link the privilege of a local process with that of the remote process.

References

[1] B. Carrier and C. Shields. A recursive session token protocol for use in computer forensics and tcp traceback. In *Proceedings of the IEEE INFOCOM 2002*, 2002.

[2] D.Dean, M. Franklin and A. Stubblefield. An algebric approach to ip traceback. In *Proceedings of the 2001 Network and Distributed System Security Symposium*, February 2001.

[3] D.X. Song and A. Perrig. Advanced and authenticated marking schemes for ip traceback. In *Proceedings of the IEEE INFOCOM 2001*, April 2001.

[4] Florian Buchholz and Clay Shields. Providing process origin information to aid in network traceback. In *Proceedings of the 2002 USENIX conference*, June 2002.

[5] H.Y Jung, H.L.Kim, Y.M.Seo, G.Choe, S.L.Min, C.S.Kim and K.Koh. Caller identification system in the internet environment. In *UNIX Security Symposium IV Proceedings.*, 1993.

[6] K. Park and H. Lee. Effectiveness of probabilistic packet marking for ip traceback under denial of service attack. In *Proceedings of the IEEE INFOCOM 2001*, April 2001.

[7] K.Yoda and H. Etoh. Finding a connection chain for tracing intruders. In *Proceedings of the 6th European Symposium on Research in Computer Security*, October 2000.

[8] F. Project. jail : Imprison process and its decendents. `http://www.freebsd.org/doc/en_US.ISO8859-1/books/developers-handbook/ja%il.html`.

[9] D. Robbins. Openssh key management: Understanding rsa/dsa authentication. `http://www-106.ibm.com/developerworks/library/l-keyc?t=gr,p=RSA-DSA`.

[10] S. Staniform-Chen and L.T. Heberlein. Holding intruders accountable on the internet. In *Proceedings of the 1995 IEEE symposium on Security and Privacy*, 1995.

[11] Spoon. Lcap : Linux kernel capability bounding set. `http://pw1.netcom.com/~spoon/lcap/`.

[12] W. R. Stevens. *UNIX Network Programming, Second Edition: Networking APIs: Sockets and XTI,*. Prentice Hall, 1998.

[13] T.W. Doeppner, P. N. Klein and A. Koyfman. Using router stampping to identify the source of ip packets. In *Proceedings of the 7th ACM Conference on Computer and communication Security*, November 2000.

[14] Y.Zhang and V. Paxson. Detecting stepping stones. In *Proceedings of the 9th USENIX Security Symposium*, August 2000.