Explaining and Recovering from Computer Break-ins

DERBI: **D**iagnosis, **E**xplanation, and **R**ecovery from Computer **B**reak-**I**ns.

Table of Contents

Explaining and Recovery from Computer Break-ins

DERBI: **D**iagnosis, **E**xplanation, and **R**ecovery from Computer **B**reak-**I**ns

Introduction

DERBI is a coordinated collection of programs that assist in *ex post facto* detection of unauthorized access and use (*intrusions*) of Unix computers. Its difference from traditional intrusion detection systems (IDSs) allows it to fulfill a need they don't address. In particular, DERBI can be used in situations where an IDS wasn't in use at the time of a suspected intrusion.

DERBI was designed based on long experience with Unix systems and with experience in tracking intrusions in these systems. Briefly, it works by closely examining the system's logs and state. For instance, DERBI notes a problem if a user was not logged in when his mail file was last read. Just as a police forensic analyst might notice latent information not easily visible, DERBI's forensic analysis looks closely for clues throughout the system.

The DERBI project was sponsored by the Defense Advanced Research Projects Agency (DARPA/ITO) from 1996 to 1999. Dr. Doug Moran was the original Principal Investigator and is responsible for much of the original design and initial implementation of DERBI. Dr. Mabry Tyson became Principal Investigator in 1998 and continued the project after Dr. Moran left for a start-up.

DERBI participated in the 1998 and 1999 Lincoln Laboratory Intrusion Detection Evaluations (IDE). Although these evaluations were designed for more traditional ID systems which used monitoring data, the test administrators provided file system data for DERBI. Overall, **DERBI did quite well in detecting attacks in the Evaluations**. Some evaluators were impressed that DERBI was able to d[illegible]t as much as it did.

DERBI – A Different Kind of Intrusion Detection System

Protecting the integrity of computer systems is a critical and ongoing requirement of system administration. This task is complex and requires a variety of tools. In the past decade, the field of intrusion detection has been the focus of much interest and research.

Much of that research has been directed towards active monitoring of systems and networks to detect ongoing intrusions. If intrusions can be detected and stopped in their nascent stages, the integrity of systems can be assured. However, while this is a critical capability, it is not the only answer to everyone's needs.

DERBI is a different kind of intrusion detection system. Rather than instrumenting a system to detect intrusions in real-time, DERBI is designed to analyze a computer's file systems after-the-fact to see if there is evidence of an intrusion. Obviously DERBI is not intended to supplant a real-time IDS as it is better to stop an intrusion before it happens. However, in a well-protected site, **DERBI could be used in conjunction with a traditional IDS**. If an attack somehow avoids or neutralizes the IDS, DERBI may be able to detect information useful in determining whether there was an attack and what happened.

In addition, DERBI was designed to be useful for those sites that do not use effective IDS systems. In particular, **DERBI would be useful for sites that do not have the resources or expertise to install, monitor, and maintain an IDS.** The tool would be useful to the local system administrator (or user, in the case of individually maintained systems) if he becomes suspicious. The tool could be run routinely or intermittently as fits the individual situation. Others, such as consultants or law enforcement officials could use the tool to help understand the state of the system.

It should be noted that DERBI can not easily detect certain classes of attacks. For instance, some denial-of-service (DoS) attacks leave no direct evidence in the file system. Indirect evidence, such as failure of mail delivery and other network activities, may indicate an event of some sort, but it would be impossible to determine whether the cause was a DoS attack or some hardware/network problem.

Comparison to Virus Detection Software

Computer users are very familiar with viruses, and any serious computer user probably protects his system from viruses. Yet many computer users use no virus protection software. Why? Some of the reasons may include:

- A lack of understanding the dangers of not being protected
- A lack of knowledge of what needs to be done to protect a system
- Procrastination
- A belief that the cost of the protection is greater than the potential loss
- A belief that the operation of the protection software interferes with the requirements for the system

One could argue that all computer users should protect themselves from viruses, but the reality is that many do not. It is usually a matter of mental economics – the user has other things he wishes to do and the effort of shopping for, acquiring, installing, and maintaining a virus protection program is too large a cost.

At least the cost seems too high until a virus hits. The economics change when the user finds that his system or data may have been damaged. The user will then scramble to fix his problem, probably by buying a virus protection/removal program. After he gets over this crisis, he may continue to run the virus protection program for a while. But as his operating system or applications evolve, he may find that his existing virus protection program is no longer adequate. He very well may slip back into the same economics of effort that led him not to be protected in the first place. The cycle repeats.

System administrators, likewise, are familiar with danger of computer intrusions. Again, the more professional system administrators have mechanisms in place to deter and detect attempts at penetration of their security mechanisms. However, many lack anything more than minimal security, say passwords and a firewall of some sort, for exactly the same reasons as above. If one considers the very many small sites at individuals homes, the number of unprotected sites is staggering. Until IDS's are part of every operating system, there will continue to be very many under-protected systems.

Unfortunately, many intrusions could be (and probably are) go undetected. But when they are detected, system administrators of unprotected sites, just as unprotected users do with viruses, will scramble to recover when they have suffered an intrusion. The administrator may not realize his system has been attacked but instead just notice some anomalies. A naïve administrator might just repair the anomalies, blaming some mysterious computer glitch, without realizing something more serious caused them. If he manages to detect that some intruder has been messing with his system, he may not know how the intruder got in or how he might block them from getting in the same way. He probably won't be able to be certain that the intruder didn't leave other ways of getting in again. All he is left with is to try to recover from any damages.

DERBI fills the niche for the situation when an unprotected system has been intruded. It can be used (whether in response to an event or as a periodic maintenance routine) to determine if evidence of an intrusion exists. As part of DERBI's inspection of the system, it checks whether the system contains known vulnerabilities.

DERBI Objectives

Assistance after a Break-In

DERBI is intended to be an aid to a System Administrator (SysAdmin) after he suspects an intrusion may have occurred. If a SysAdmin suspects an intrusion, he is probably under considerable pressure to analyze the situation, understand the compromises, losses, and other damages to the system, and to restore the system to a secure status where the intrusion won't happen again.

Under this kind of time pressure, a SysAdmin can't easily update his skills to become a security expert. He can't spend too much time manually analyzing the system to try to determine what happened. For a variety of reasons (including time and cost), he may be unable to hire security consultants to help him with his crisis. What he would appreciate is a security consultant in a box that would analyze his system for him. DERBI is one step along the road to this capability.

No Prior Set-up Required

In keeping with the concept of an IDS for unprepared sites, one of the desiderata in the design of DERBI was that DERBI should require nothing more than what is standard operating procedure for a particular operating system (OS). That is, DERBI should be able to function with an OS installed just as the vendor delivered it. DERBI *can* take advantage of other features that may have been added to the system (such as TCP Wrappers) but these are not required.

Explanation and Guidance for Recovery

To aid the SysAdmin, DERBI provides an explanation of the evidence it finds of an intrusion and related anomalous evidence in the system. The SysAdmin may review this information and determine what course of action to follow. Among the evidence presented are indications of known vulnerabilities that exist on the site.

These reports might also be used for gathering data about attacks across a larger number of sites.

How Effective is After-the-Fact Intrusion Detection?

One of the objectives of the DERBI project was to see whether *ex post facto* intrusion detection could detect a wide variety of attacks. After all, our only sensor is a post-mortem forensic analysis of the file system. The cracker has full opportunity to clean this up but that is often limited to obliterating certain logging information.

The Intrusion Detection Evaluations indicate that DERBI is effective in the case that no active IDS was in place.

One of the tenets of our system is that if an intruder gets into a system, he is likely to poke around in the system. Perhaps he wants to find alternative vulnerabilities; perhaps he wants to install a Trojan horse or data collection program (sniffer); or perhaps he is modifying or looking at files. We feel that a strength of DERBI is that **even if he uses some new exploit, we will detect his subsequent activity**. Almost by definition, this subsequent activity will modify the file system and therefore may be available to our sensor. Attempts of a cracker to camouflage his existence may actually reveal his presence as he keeps modifying the file system (by accessing files).

DERBI Usage

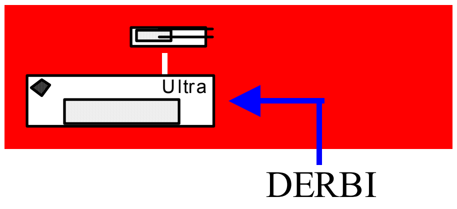
DERBI could be deployed in any of several modes to analyze a suspect system.

Direct Installation on Suspect System

DERBI could be installed directly on a suspect system, using its CPU for executing DERBI. This has the advantage that DERBI has full access to the local environment (users, mount points, time of day, etc.). For a small site, this may be the only available option.

However, this deployment is not advisable as the execution of DERBI is dependent upon the suspect system. An intruder could manipulate the system to cause DERBI to fail to notice anomalies.

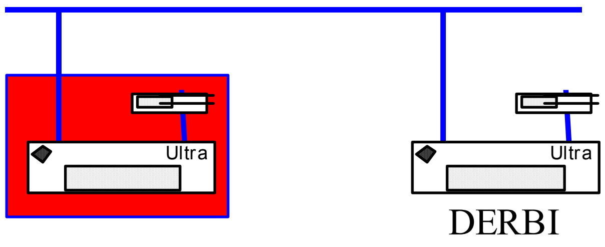
This may be the preferred configuration for periodic runs of DERBI, searching for unnoticed intrusions.



Inspection of Suspect System from a Trusted System

DERBI could be installed on a trusted system in the same site as the suspect system. This has the advantage that the operation of DERBI is secure. Some of the environment is probably shared by the two systems, but DERBI must be capable of recognizing the differences in the environment between the suspect and trusted systems. The trusted machine must have privileges to access all the data on the suspect machine.

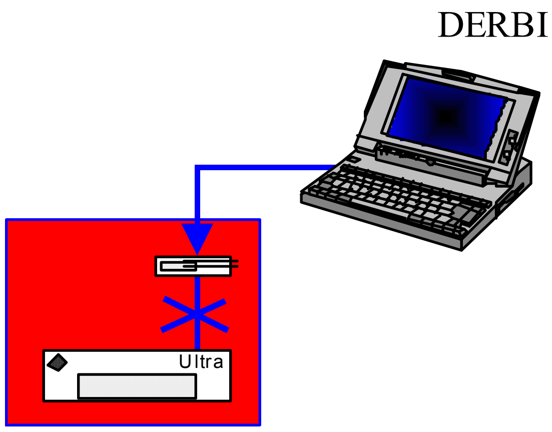
This configuration is preferred over the execution of DERBI on the suspect machine.



Analysis by a Portable DERBI Machine

A portable computer (perhaps belonging to a security consultant) running DERBI could be attached to the file systems of a suspect system. The file systems would be available as data to the DERBI system. DERBI needs to understand more of the system, such as mount points, users, and user directories.

In this configuration, the suspect machine is taken down and the disks mounted instead to the portable machine. This assumes that hardware issues are not too much a problem.



Inspection of System Snapshot

The preferred use of DERBI on a suspect system involves taking a snapshot (backup) of the file system by `ufsdump` and creating a new file system containing a copy of that snapshot. This data can then be analyzed without fear of corrupting the data any further. Any repair or recovery of the suspect system can be carried out in parallel.

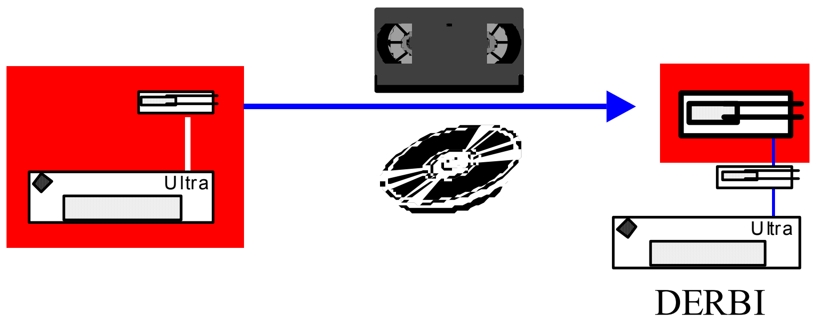
Taking a full dump of the file systems may be time consuming. However, if civil or criminal prosecution of an intruder is a possibility, the collection of a system snapshot may be necessary to establish the state of the machine, log files, or other evidence.

If there is a regularly scheduled backup of the system, these backups, plus a newly-run incremental backup, can replace the full snapshot. Note that an intruder can modify files so that a newly modified file may appear to be old and so not making it to this incremental backup. This is a risk that must be weighed against the cost of taking a full snapshot.

If for some reason an intrusion isn't noticed for quite some time, then a regular backup soon after the intrusion may reveal details that have been trampled over in the intervening time. DERBI could analyze the state of the system at the time of that backup.

For the IDE, we provided a routine that gathered much of the data DERBI needed from the file system without the need to collect the contents of all the files. While this worked for the most part, there was some data we failed to collect which could have resulted in improved performance. (In a real intrusion situation, you can't predict ahead of time what files might contain intruder data or programs, so it is better to get dumps.)

For DERBI, we modified the restore program for Solaris, `ufsrestore`, to accurately restore the creation time on files and all the times on directories. The standard `ufsrestore` does not attempt to set these properly. In order to use this deployment, these times need to be set as they were on the original file system. An alternative approach would be to retrieve the information directly from the backup media without doing a restoration.



DERBI Structure

Head-Body-Feet

DERBI can be thought of as one body with three main parts. The multiple “feet” do the walking over the (possibly remote) file systems collecting data; the “head” does the analysis; the “body” ties the two together. The head directs what data should be gathered or checked. The body translates that into commands to the various feet routines. The feet routines gather the data and report it back to the body. The body then formats the data in an appropriate fashion to respond to the query from the head.

The feet gather the evidence which the head then analyzes and determines the belief of various attacks. The feet portions of the system are designed to run on remote machines and are typically coded in PERL or C. The body directs the feet and annotates the results to indicate what machine reported the results. It is also responsible for canonicalizing such information (e.g., for time differences, for mount point differences, etc).

Head

The “brain” behind the head is PRS (Procedural Reasoning System). Its knowledge consists of a collection of evidence schema along with a networked representation of how these bits of potential evidence lead to more general attack patterns. Each of the evidence schemas indicates a kind of evidence that may indicate a kind of attack and how likely the attack is based on this evidence. Each of the evidence schemas is a stand-alone document. A system may be updated for new attacks by the addition of new schema.

Figure 1 shows an evidence schema for a buffer overflow attack on the eject command. The FILE in question is `/usr/bin/eject` (or wherever soft links lead you to). In determining the applicability of the EVIDENCE, the system first checks 1) Is the particular version of `eject` on the system is known to be vulnerable and 2) Was it last run (accessed, actually) during the timeframe of interest (window of opportunity, the time during which the intrusion is thought to have taken place). Otherwise, the belief is set to 0 that this command was improperly used.

If the executable was last accessed more recently than either of the devices that it is normally used for, then the belief that it was used in an attack is set to 40%.

In this case, the system asserts (POSITs) into its knowledge database the fact that the intruder may have gotten a shell with root permissions at the time of the access.

Finally, an explanation is generated for this bit of evidence that explains how a root shell may have been exploited at that time.

Part of the operation of the head is to correlate all the evidence by time and to present them to the user. A segment of an output from an evaluation is seen in Figure 2. This time-oriented presentation of the evidence is very effective. The operator can see the progression of events as an intruder

1. tries to gain access to the system,
2. gets access,
3. prepares for gaining privileges,
4. takes advantage of an exploit to get privileges,
5. uses the privileges to gain access to (or create) files,
6. and then finally exit the system.

```
EVIDENCE-TYPE       (exploit (   setuid root) buffer-overflow)
UNIQUE- NAME        eject-1
EVALUATION-  NAME   eject
PATHS             (follow-links '("/usr/bin/eject"))
EVIDENCE
( ((not (and    ;; vu lne rab le co mmand
            (   command-version-vulnerable-p DIR FILE)
              ;; used in interv al o f interest
            (   window-of-opportunity (     TimeAccessed PATH))))
    0   0)  ;; a ssign 0% probab ility to co mmand be ingu sed
            ;; and  0% be lieve  tha t it wa s
  (( greater-than (   TimeAccessed PATH)
           ;; u se is later than exp ected effects
       (  max ( TimeModified "/   cdrom") (  TimeModified "/floppy")))
    40 100)   )  ;; 40 % probab ilityo f exploit,
                 ;; no  change   in be lief abou t whe ther it was exp loited
POSIT
((posit ((TIME (   TimeAccessed PATH)))
        (  compromised-shell "root" TIME *unknown-time*)))
EXPLANATION
(explain-evidence
          (        ;; va riable dec larations
               PATH
               (TIME (print-      unix-time (   TimeAccessed PATH)))
               (TIME2 (print-      unix-time (  TimeModified "/   cdrom")))
               (TIME3 (print-      unix-time (  TimeModified "/floppy"))     ) )
          (TimeAccessed    PATH)  ;; Òas-ofÓ time
  "The command ~S is version vulnerable to a buffer overflow attack
                 and appears to have been used at time ~A
                 which is more recent than two associated files:
               /   cdrom (~A) and /floppy (~A)."
       PATH TIME TIME2 TIME3)
```

Figure 1: Evidence Schema: EJECT Buffer Overflow

One of the unusual situations in the evaluations was that not all of these elements were present. In particular, often the exploit happened, but the intruder did nothing with it. Certainly some intruders might act this way, but many would not. It is this subsequent activity, not the exploit, that DERBI is most sensitive to.

PRS can support arbitrary queries in the EVIDENCE field. In particular, the system can ask for operator intervention. It can ask for the operator for information ("Is it likely that user Doe would login from cracker.ru?" "Did an operator access Doe's mail files yesterday"). PRS can also support asking the operator to execute code such as restoring a file from backup.

```
+04:53:25 later
====================================
Time: 23-Jul-   1998  14:32:39 EDT (901218759)
Exploit: Suspicious-login (Suspicious-login)

Login for user "   darleentÓ from host 194.7.248.153
-------------------------------------------------------------
+00:00:12 later
====================================
Time: 23-Jul-   1998  14:32:51 EDT (901218771)
Exploit: DOWNLOADING-EXPLOIT (UUDECODE-1)

"/usr/bin/uudecode" is often used by crackers and
rarely by users, and appears to have been used at
time 23-Jul-1998  14:32:51 EDT.
-------------------------------------------------------------
+00:00:23 later
====================================
Time: 23-Jul-   1998  14:33:14 EDT (901218794)
Exploit: EJECT (EJECT-1)

The command "/usr/bin/eject" is version
vulnerable to a buffer overflow attack and appears
to have been used at time
    23-Jul-  1998  14:33:14 EDT
which is more recent than two associated files:
/cdrom (12-Feb-   1998  15:42:46 EST)
and
/floppy (20-Jul-   1998  10:32:15 EDT).
Asserting belief/plausibility = (40 100)
------------------------------------------------------------
+12:10:32 later
```

Figure 2: Sample Segment of Output

These POSIT'ed data from the schema are inputs to the analysis by the PRS system. DERBI's PRS knowledge base includes a graph representation of how these bits of information can be combined into evidence of a pattern of an attack**. DERBI is designed to detect the overall attack, not just exploits**.

Figure 3 is a portion of this graph representation for DERBI before any evidence is acquired. The square shapes represent the posited information from the evidence schemas. The yellow lines represent the flow of information from the evidence nodes to conclusions represented by ovals. Intermediate conclusions are collected into more general conclusions. Intermediate conclusions include rather specific aggregates such as Buffer Overflow, Clock Reset, SymLink attack, etc. The most general conclusions are Root-Compromise, User Compromised, Camouflage, and Subsequent-Activity.

Disparate pieces of evidence can give weak evidence for several of the evidentiary trees. The combination of these bits of evidence (by a Dempster-Shafer method) can provide evidence of higher conclusions. When the correlation of evidence provides a sufficiently high belief (loosely, probability), the system indicates the result by turning the green conclusions red.

Any end-user is not expected to understand the PRS graph of evidence dependencies. Instead, when DERBI reaches a conclusion, DERBI will give the user an explanation of how these bits of evidence are combined into an overall number indicating the belief in the top-level conclusions. These explanations are in addition to the explanations of each of the individual pieces of evidence.

The numeric beliefs in the evidence schema and in the PRS graph are rather arbitrary and based upon the developers' experience. We believe that the numbers are not critical and that the design will result in a system whose top-level beliefs will tend towards either 0% or 100%.

Feet

The data collection is done by DERBI's feet. DERBI initially does a sweep of the entire file system gathering the basic information about the files: name, owner, protection, date of creation, date of modification, date of access, size and other information in the inode. In particular, DERBI is very careful to collect the access times on files and directories before disturbing them.

Other system information is also gathered and utilized, including system name, mount points, users and other information from `/etc/passwd`, groups, crontabs etc. User dot files (for example, `.cshrc` and `.history`) are collected for examination. System executables are checksummed to compare against tables of known good (or vulnerable) versions. System log files are collected for analysis (see Figure 4: Relations of Log Files for some of the log files used).

Strictly speaking, the feet don't include any of the processing of this data. However, what the capabilities programmed into the feet are aligned with the kind of data needed by the head. Much of the information is processed to extract just the information we need. For instance, a catalog of symbolic links is built in order to follow those more quickly when we need them.

Reserved for an imageo f PRSg raph

Figure 3: PRS Graph of Evidence Relations

Data Collection

Obvious Data

DERBI searches the suspect system for many clues at what happened when. In doing so, we don't overlook the obvious. If files exist bearing the names of known cracker programs, DERBI detects and reports these. If root suid files exist outside known system directories, these are reported. If empty or crackable passwords[1] exist or improper protections exist on critical system files, the user is alerted and the system uses this as evidence that an intrusion may have occurred at any time.

These are just a few of this class of data. Obviously not all of these are sure-fire indications of an intrusion, but in aggregate, and if temporally near other events, these may indicate intrusions.

Subtle Data

DERBI also searches for more subtle data. If a vulnerable and privileged suid program has been executed, there is some low level of belief that an intrusion may be related. Even a program like `automount` which is often called by the system causes this alert. If the last execution of a vulnerable program aligns with other suspicious events, it may provide evidence of the exploited vulnerability. If it doesn't align, the event is spurious and can be ignored.

If the system detects that a program was accessed but evidence exists that it wasn't run in a standard mode, the belief in an intrusion jumps significantly. For instance, if a vulnerable `format` was accessed, we presume it was run. If neither the cdrom nor floppy device was executed, we are highly suspicious that the `format` vulnerability was exploited.

We found it very useful to compare file access and modification/creation times against the times of activity of those users legitimately able to read or write those files. A cracker may hide his actions from various monitoring systems (for instance, by installing a modified `ps`) but if he touches files of an idle user, he has left his calling card. This is also sensitive to the case in which we see the system after system logs have been corrupted and there are a number of users that are legitimately on the system and accessing their own files, yet no record of their login occurs. These two cases can usually be distinguished by other patterns of file accesses (such as access of dot-files such as `.cshrc` and other ordinary activity such as reading or sending mail or the creation of browser cache files during a login). In either case, an intrusion has occurred; the only question is, what does the evidence indicate.

[1] The `crack` program is run by DERBI against the set of passwords to determine these.

Confounding this kind of information is normal system activity. Normal system activity (not including operator activity) doesn't access or create too many files owned by normal users. A legitimate operator may be accessing users' files as part of his duties. However, the operator will have left a clear audit trail of his become root.

Other examples of subtle data are the relationships between mail delivery/acceptance and the network. In a normal system, the arrival of mail is somewhat like background radiation. It is happening all the time but rather erratically. If mail quits arriving, something strange is happening.

We paid quite a bit of attention to file access/modification/creation times. If a file was created but the directory wasn't modified more recently, this was a sign of something slightly suspicious. If system files had times that were on the minute boundary, this was slightly suspicious (cracker tools may not bother to allow him to set it to anything but the minute such as you see in a `ls -l`).

User logins are tracked. If a user logs in from a new host (for him) or a host that has become suspicious, a warning is issued. Again, this is only weak evidence of an intrusion. But, if this aligns with other bits of evidence, the event becomes more suspicious.

Again, DERBI detects quite a few subtle pieces of evidence.

Obscure Data

As DERBI was developed, we found more rather obscure and idiosyncratic relationships to use.

Log files often contained obvious data about exploits (for instance, messages about multiple failed logins). By comparing the information in multiple log files (see Figure 4), we were able to view subtle data where an intruder might not properly cover all his trails. But we also found obscure evidence of attempted telnet logins that never actually got logged (and so there was no reason to cover them up). The data structures in the log files reflect the connection, but since no login succeeded, no log entry is ever reported.

Other obscure data we found was that deleted file names remain almost intact for some time in directories. Thus even if the attacker might delete his files, we may be able to recover some information about the files he used. (The deleted names lose their first letter.)

We found other obscure relationships in the ownership of the pseudo-devices and their modification times. This seems to give an alternative way to determine, in some cases, who logged in to what pseudo-device and when they logged in. When the system logs have been tampered with, this information was sometimes able to provide hints at what happened. Unfortunately, there were some cases in which the IDE data seemed anomalous so this gave us some unexpected problems. We're not sure whether this is a trait of the operating system or whether the IDE somehow accidentally corrupted this data (see the section on evaluation).

Obscure data tends to be idiosyncratic and might only be valid on a particular version of an operating system.

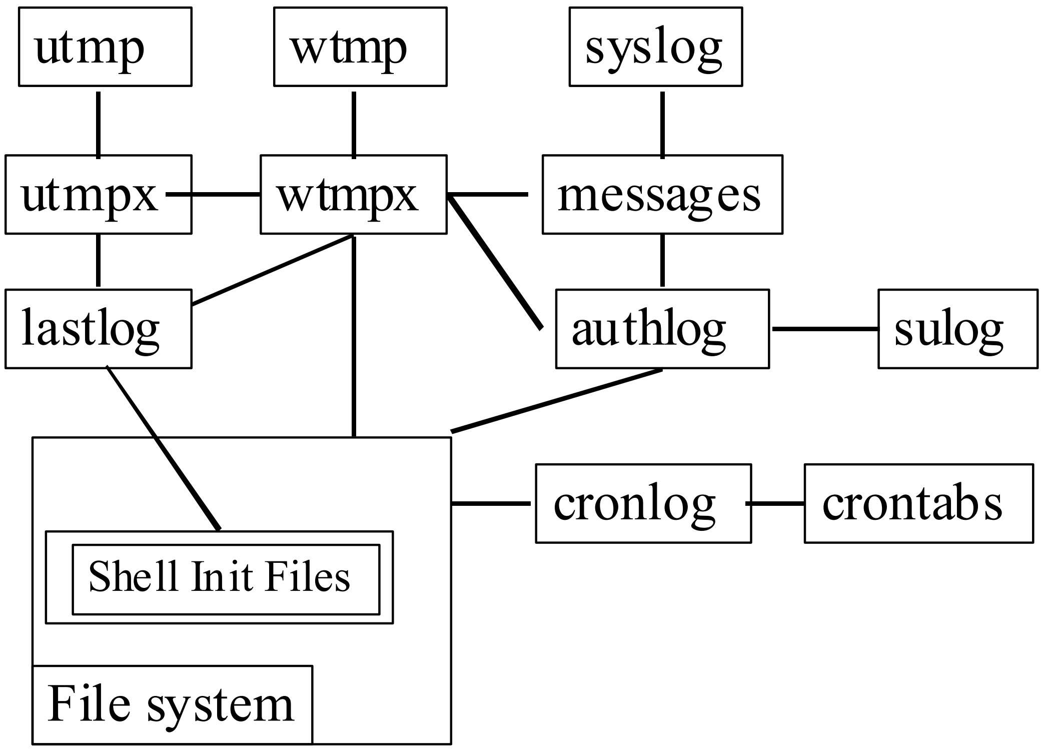


Figure 4: Relations of Log Files

Evaluation

Lincoln Labs IDE

DERBI participated in the 1998 and 1999 Lincoln Laboratory Intrusion Detection Evaluations (IDE) (http://ideval.ll.mit.edu/). These evaluations were sponsored by Defense Advanced Research Projects Agency (DARPA ITO) and Air Force Research Laboratory (AFRL/SNHS).

The evaluation consisted of data sets of sensor (BSM data, audit data, sniffer data, etc. and file system dumps (1998) and DERBI-collections of file system data (1999)) covering several weeks of simulated normal system operation of a test site in which specific intrusions occurred. The data sets were divided into training data and test data. Training data was presented well in advance of the test to allow participants to understand the format of the data, the configuration of the test site, and the required format for results.

These evaluations measure probability of detection and probability of false-alarm for each system under test. The actual results of the systems' performances are not available for publication. These are research systems and the evaluation is also an initial foray into IDS evaluation. However, we will discuss DERBI's performance and how it related to the overall performance of the other systems.

1998 IDE

The IDE in 1998 was the first attempt at evaluating the performance of IDSs. DARPA had pursued evaluations in other research areas and desired to do the same for the computer security research area. The task was daunting as the collection of data for these evaluations would be difficult. The evaluation methodology was also a sensitive area. Six research groups participated in the offline evaluations in the fall of 1998.

The design of the evaluation was clearly done with real-time IDS systems in mind. Some kinds of attacks, such as DoS, are not what most system administrators would consider intrusions, and are not generally detectable in the file system. A class of attacks was considered "stealthy" but the attempts at stealth were of no difference to DERBI. DERBI's requirement of a dump of the file system had apparently not been in the original plans. We are grateful that the evaluators were flexible enough to accommodate DERBI.

Overall, when compared to a commercial IDS keyword system, the research systems had vastly better results with better detection and fewer false alarms. However, no commercial IDS vendor participated in these evaluations so it would be unfair to give absolute credence to that result.

Each system was measured only against the kinds of attacks it had sensors that might detect the attack. Thus a system, such as DERBI, that only ran on Solaris machines would not be held accountable for attacks against SunOS machines. As a corollary, DERBI was only tested against a relatively few number of attacks.

DERBI detected approximately 55% of the attacks ("Attack Scores") with a false positive rate of less than one a day. We detected about 67% of the known ("old") attacks and 40% of the novel ("new") attacks. (These numbers for the evaluations should be considered as only rough indications of relative performance. There is no evidence that these numbers would be appropriate for a fielded system.)

DERBI's performance was among the best of the 1998 participants. Even so, we note that some of the attacks we were unable to detect, specifically DoS attacks, simply are not visible to our sensor. As one would predict, DERBI did better in detecting User-to-Root attacks than Remote-to-Local (login) attacks. The former often leaves more evidence in the file system.

1999 IDE

In 1999, the simulated site was more complex with further attack classes. A major emphasis was the need to identify novel attacks. Seven research groups participated in the evaluation, submitting results from a total of 17 differently configured systems.

In 1998 the evaluators felt that taking full dumps of the system after each day's simulation was too labor-intensive. We provided Lincoln Labs with a routine to capture much of the file information we needed for DERBI to analyze the system.

Unfortunately, we failed to specify the need to capture `crontab` files. During evaluation, we were unable to detect the existence of these jobs (during which a legitimate user process may access the file system but without a corresponding login).

Again, DERBI was only measured against attacks on Solaris machines. We had ported the feet portion of DERBI to Linux during the summer but forgot to supply to Lincoln Labs a file-system information collection program for Linux. As a result, no data for DERBI was generated on Linux systems during the simulation.

Another testing artifact complicated the evaluation of DERBI. After Lincoln Labs ran a day's simulation, they might not run the next day's simulation right away. Instead, they sometimes left the system running while they performed various tasks on it. For instance, if after they had run Monday's simulation, they then did tests or whatever on the system, the system's clock would continue to run into Tuesday. In some cases, the interval between simulations might be two or more days. Finally, when they began Tuesday's simulation, they would turn the clock back to Tuesday morning.

This works well for the real-time audit data, but the file system data has been corrupted. When the file system data is collected on Tuesday night, there would be information that files were created or accessed during that "Tuesday" that happened between the simulation, as well as the intended file system changes that happened during the simulated Tuesday. In some cases, the data on Tuesday night indicated changes to the file system with Wednesday's date! We were able to ignore the Wednesday dates, but there was no easy way to distinguish file system changes that happened between simulations versus those that happened during a simulation.

Still, **DERBI again did well despite the fact that the intrusions were now more complex**. 61% of all attacks were detected. 62% of the novel attacks were detected. The number of false alarms was up (2.5/day) due primarily to the problem of duplicated times. Again, DERBI was among the top performing systems. The pattern held of DERBI performing better on the User-to-Root that Remote-to-Local. In a new class of attacks (DATA) where the IDS was to detect unauthorized access to files, DERBI detected more than 80% of the attacks.

These numbers for the evaluations should be considered as only rough indications of relative performance. The increased complexity of the intrusions in the test from 1998 to 1999 significantly affected these numbers. There is no evidence that these numbers would be appropriate for a fielded system.

Benefits of Evaluation

These evaluations were extremely useful to the development of DERBI. The primary benefit was that the evaluations provided sets of intrusion data that we could not easily have acquired or produced. Rather than having each research group generate pseudo-realistic data for testing, the effort was concentrated in one group who then provided it to the participants.

Secondly, the joint evaluation provides each research group and the funding agencies a better understanding of how each technology compares. By forcing the research community into a joint test, we have some basis for this comparison. However since the various systems address different aspects of intrusion detection, there are few completely comparable systems.

Finally, the evaluations focussed the development effort towards near-real-world situations. This is both a benefit and a restriction. In order to perform well on these tests, considerable effort had to be expended on the tests themselves. This took away from research efforts in other directions. For DERBI, this meant less time was spent in developing the system as a tool for inexpert SysAdmins by researching better explanation and recovery capabilities. Instead we improved the detection portion of DERBI.

Porting to New Environments

Linux Port

DERBI was developed to run under Solaris. During the summer and fall of 1999, portions (the “feet”) of DERBI were ported to run under Linux. The port did not go quite as smoothly as one would hope.

When we initially began our Linux port, we aimed for the then current version of Redhat Linux which was 6.0. When we realized that the IDE systems were only running Redhat 5.0, we had to reconfigure our Linux system for that version. Unfortunately this was complicated by the fact that our hardware was too new for all the drivers to exist in Redhat 5.

We also found there were differences in some of the details of logging between the two systems. Redhat 5 didn’t have all the information available in Redhat 6.

Conversion to Linux required going over all of the system. Assumptions built into the software needed to be reviewed and revalidated. Log files needed to be examined to determine how messages were recorded slightly differently. Obviously some of the obscure data we were able to use had no parallel in Linux. We weren’t as intimately familiar with Linux to find Linux-specific obscure relationships.

When we had most of the pieces running, we realized that we were not going to be able to participate in the IDE on the Linux systems. In response to Lincoln Labs’ request, Doug Moran had provided them with a tool to extract the information from the file system that DERBI needed. But this was done before the Linux conversion began. Dr. Moran had left the project by the time we began the conversion and we didn’t realize until the data came out that the only data available for DERBI would be on the Solaris systems.

As a result, the Linux port was never tested by the evaluation mechanism.

Porting the system gave us a healthy respect for the need for the abilities of an expert in the target OS. For this port, we had one of the developers of FreeBSD, but even he was unable to quickly port the system. (But I expect that FreeBSD development will be more conscious of the need for thorough logging!)

Complex Sites

Another area that we had wanted to pursue was to explore the difficulties in working on a complex site. In such a site, a single file can be accessed or modified from any of a set of computers. A file may be known by different names on different machines. The clocks may be different on the different machines (so whose time is used to set the access time on files?). Each machine may have different sets of users, or the same user may have differing uids (while that might not be desirable, a cracker may cause that situation). Exploits could be spread over different systems in novel ways.

The issues here are not just for a DERBI style system. Other IDSs will need to face some or all of these problems.

Summary

Weaknesses

DERBI's primary feature is both a strength and a weakness is that it isn't an alarm system, watching everything happening on the system, and ready to sound an alarm the instant something unusual happens. As a result, it can't do anything about intrusions until the SysAdmin notices something unusual.

DERBI's only source of information is the file system. As a result, it is dependent upon whatever the operating system chooses to record in log files or elsewhere. On an operating system without access dates, DERBI would be in many ways partially blinded.

The fact that an intruder has access to the file system means that he has the opportunity to obliterate or confound DERBI by manipulating log files and other information. For instance, if an intruder with root access causes all the files in the system to be accessed, DERBI can't tell too much about what he did.

DERBI is also vulnerable to other users stepping on the data in the file system. If the interval between the intrusion and the application of DERBI is too long, some of the information may be lost. A second intruder may also confuse the situation and possibly completely hiding the fact of a prior intruder.

We've used Sun's `ufsdump` program to record file system information. This program is unusual in that it can record the file system information without significantly modifying it. Other backup programs are different and will modify either the creation time or access time of files they back up.

As with all IDSs, if a cracker knows all the triggers you use to detect him, he can be more effective at avoiding them. In response to a query as to whether we felt someone could get into and out of a system without being detectable, we felt the answer was “No”. However, one could hide much of what one did.

Strengths

Again, DERBI’s primary feature is both a strength and a weakness. Rather than burdening down a system with instrumentation, DERBI can check on the system when it is needed or on a regular system at off-hours.

Evaluations indicated that DERBI did a good job at detecting intrusions. We feel that in the real world, it would do even better.

DERBI uses a variety of redundant information to confirm that the system is consistent. Inconsistencies by themselves are suspicious. Much of this redundant information is relatively hidden so crackers are less likely to make sure all is consistent.

DERBI is designed to be usable by a SysAdmin that is not a computer-security expert who is under pressure to investigate, resolve, and recover his system in the least amount of time.

DERBI can be run from any of several configurations depending upon the resources available. These range from the most convenient (single computer) to the most reliable (examining a file system rebuilt from a backup of the suspect system). The operator can choose what best fits his needs.

Future

One of the issues that comes up with DERBI and many of the other IDSs is the issue of privacy. We ran into this issue during the 1999 IDE. We considered looking at browser cache’s in much the same way as one would look at telnet logs. If a user accesses a cracker site via the web, his account begins to look suspicious. It doesn’t take much imagination to expand the set of red-flag sites until you get to the point at which individuals may have claims of privacy.

One of the most pressing questions is whether DERBI and its technology are relevant. Obviously the problem of intrusions won’t go away. It is clear that a system that detects intrusions in real time is a preferable tool at a site that can afford to have that. The question is then whether these tools can be made easy enough to install and maintain on each and every computer.[2] If not, then DERBI has a utility. Even if so, DERBI’s technology can be used to help investigate intrusions that avoid detection by real-time IDS.

[2] It is our belief that network sniffing tools will be totally ineffective when most network traffic is encrypted. As a result, host-based IDS is seemingly the only choice.

Dr. Doug Moran, the original designer of DERBI, is currently part of a start-up working on the problems of computer security. It seems likely that some of DERBI's technology may be involved in future commercial products.

A DERBI-technology system could be developed for other operating systems, most notably Windows NT. The system would need to be rewritten basically from scratch. There are a number of issues that need to be considered, including logging and the difference between primarily multi-user vs. primarily single-user systems.

Commercialization of a DERBI-technology system would require a plan to solve the field-update problem. The system would need to be able to updated in much the same way that current virus checking programs are updated.

Summary

DERBI began as an effort to build a system that performed like a computer security expert to assist a SysAdmin in understanding intrusions. The prototype system performed quite well on both the 1998 and 1999 Intrusion Detection Evaluations. The research project DERBI has proved its primary objectives – an after-the-fact IDS can effectively detect and explain computer break-ins.