Integrating IDS Alert Correlation and OS-Level Dependency Tracking

Yan Zhai, Peng Ning, Jun Xu
Cyber Defense Laboratory
Department of Computer Science
North Carolina State University
Raleigh, NC 29695-8207
{yzhai, pning, junxu}@ncsu.edu

Abstract

Intrusion alert correlation techniques correlate alerts into meaningful groups or attack scenarios for the ease to understand by human analysts. However, the performance of correlation is undermined by the imperfectness of intrusion detection techniques. Falsely correlated alerts can be misleading to analysis. This paper presents a practical technique to improve alert correlation by integrating alert correlation techniques with OS-level object dependency tracking. With the support of more detailed and precise information from OS-level event logs, higher accuracy in alert correlation can be achieved. The paper also discusses the application of such integration in improving the accuracy of hypotheses about possibly missed attacks while reducing the complexity of the hypothesizing process. A series of experiments are performed to evaluate the effectiveness of the method, and the results demonstrate significant improvements on correlation results with the proposed technique.

1 Introduction

Intrusion detection has received a lot of attention in the past two decades. However, current intrusion detection systems (IDSs) often generate huge numbers of alerts as well as numerous false positives and false negatives. These problems make the reports from IDSs very hard for security administrators to understand and manage. Many researchers and vendors have proposed various alert correlation techniques (e.g., [6, 7, 17]) to make large numbers of IDS alerts more understandable and at the same time reduce the impact of false positives and false negatives.

Recent alert correlation techniques can be divided into three categories: similarity-based correlation (e.g., [3, 5, 15, 17]), correlation by matching with pre-defined attack scenarios (e.g., [6, 7]), and correlation based on the prerequisite (also called preconditions) and consequence (also called postconditions) of individual attacks (e.g., [4, 10]). Each technique has its advantages and disadvantages. However, the correctness of correlation results is strongly affected by the false positives and false negatives among IDS alerts.

Several researchers recently investigated integrating additional information sources into alert correlation to improve its quality. In [9], a formal model named M2D2 was proposed to represent data relevant to alert correlation. The technique in [12] reasons about the relevancy of alerts by fusing alerts with the targets' topology and vulnerabilities, and ranks alerts based on their relationships with critical resources and users' interests. In [18] a statistical reasoning framework is proposed to combine IDS alert correlation with local system state information from tools such as system scanners and system monitors. These approaches can improve the performance of correlation by integrating different sources of security-related information. However, the correlation results are still not yet satisfactory. For example, neither of the approaches can deal with false negatives quite well.

Thus, it is desirable to find additional ways to integrate other information sources to further improve alert correlation.

In this paper, we propose to harness OS-level event logging and dependency tracking to improve the accuracy of alert correlation. OS-level dependency tracking is a recently developed technique to analyze the system operation history toward a given object. It tracks dependency-causing events such as process forking and file operations in the system event log, and spans up a tree of system objects connected by these events from the target object. Though a very useful tool for forensics applications, backtracker has two limitations. Firstly, because it is system call oriented, the complexity of tracking and tracking results can be very high. For example, during a normal system run, the resulting dependency graph of such a tracking (using *Backtracker* [8]) can contain up to tens of thousands of objects and hundreds of thousands of edges. Such complexity with tracking is obviously time and resource consuming, while the tracking results are also hard to understand. Secondly, the tracking is highly dependent on the availability of so-called "detection points", which are significant evidence of system being attacked. However, such "detection points" are not usually available, making it inconvenient to use in security administration. Integrating event logging and dependency tracking tools with alert correlation can potentially address the above limitations, and at the same time improve the performance of alert correlation.

Our integration method is based on the following observations: Firstly, most attacks have corresponding operations on specific OS-level objects. Secondly, other than a few exceptions, if one attack prepares for another, the later attack's corresponding operations would be dependent on the earlier one's. Because logging these system calls is more straightforward than detecting attacks using rules and signatures, such information is considerably more accurate and trustworthy than the IDS alerts. Utilizing such information will improve the performance of alert correlation.

Given the alert correlation results and an OS-level dependency tracking tool, the integration is done in two phases. The first phase is to identify the system objects corresponding to the IDS alerts based on their semantics. The second phase is to verify the relationships demonstrated in the correlation result or discover the missed relationships among the alerts by tracking the dependencies among their corresponding system objects using a dependency tracking tool such as Backtracker.

The contribution of this paper is the development of a framework to integrate the information from OS-level event logging and dependency tracking into IDS alert correlation. With the support of OS-level event logs, we can achieve better accuracy in the final result than the original alert correlation method. We also discuss how such integration can facilitate the hypotheses about possibly missed attacks. Finally, we evaluate the effectiveness of this scheme by performing a series of experiments. Our experiment results show that the integration can greatly improve the correctness of correlation and help making hypotheses about possibly missed attacks. For example, in our experiment, our approach can totally remove the false correlations in all three attack scenarios.

The remainder of this paper is structured as below. Section 2 briefly introduces the background of alert correlation and OS-level dependency tracking. Section 3 discusses the details on how to integrate OS-level object dependency tracking into alert correlation. Section 4 gives experimental results used to evaluate our approach. Section 5 discusses related work. Section 6 concludes and points out some future directions.

2 Background

As discussed earlier, our goal is to improve intrusion alert analysis by integrating OS-level event logging and object dependency tracking into IDS alert correlation. In this section, we give a brief introduction to the alert correlation and OS-level dependency tracking techniques to be used in our method. For alert correlation, we use the method based on attack's prerequisite and consequence [10], due to the ease to make connections between alert correlation and OS-level objects in this method. The dependency tracking technique used in this paper is *Backtracker* [8], which to our best

knowledge is the only such tool available.

2.1 Alert Correlation Based on Prerequisites and Consequences of Attacks

Here we give a brief overview of the alert correlation method in [10]. This method correlates intrusion alerts using the prerequisites and consequences of attacks. Intuitively, the prerequisite of an attack is the necessary condition for the attack to be successful. For example, the existence of a vulnerable service is the prerequisite of a remote buffer overflow attack against the service. The consequence of an attack is the possible outcome of the attack. For example, gaining the root access from a remote machine may be the consequence of a ftp buffer overflow attack. In a series of attacks where earlier ones are launched to prepare for later ones, there are usually connections between the consequences of the earlier attacks and the prerequisites of the later ones. Accordingly, we identify the prerequisites (e.g., existence of vulnerable services) and the consequences (e.g., gain certain privilege) of attacks, and correlate the detected attacks (i.e., alerts) by matching the consequences of previous alerts to the prerequisites of later ones.

The correlation method uses logical formulas, which are logical combinations of predicates, to represent the prerequisites and consequences of attacks. The correlation model represents the attributes, prerequisites, and consequences of known attacks as alert types. The correlation process is to identify the *prepare-for* relations between alerts, which is done with the help of prerequisite sets and expanded consequence sets of alerts. Given an alert, its prerequisite set is the set of all predicates in the its prerequisite, and its expanded consequence set is the set of all predicates in or implied by its consequence. An earlier alert t_1 *prepares for* a later alert t_2 if the expanded consequence set of t_1 and the prerequisite set of t_2 share some common predicates. Intuitively, an earlier alert prepares for a later one if the consequence of the earlier one "contributes" to the prerequisite of the later one. An alert correlation graph is used to represent a set of correlated alerts. An alert correlation graph $CG = (N, E)$ is a connected directed acyclic graph, where N is a set of alerts, and for each pair $n_1, n_2 \in N$, there is a directed edge from n_1 to n_2 in E if and only if n_1 prepares for n_2.

The advantage of this method is that the correlation result is easy to understand and directly reflects the possible attack scenarios. However, as the correlation is solely based on IDS alerts, the result highly depends on the quality of the IDS alerts. For example, the result may contain false correlations when there are false alerts.

2.2 Backtracker

Backtracker is an OS-level dependency tracking tool [8]. Backtracker monitors specific types of OS-level objects, i.e., processes and files. The objects are kept in a log with their properties such as the uid of the objects. It also monitors specific dependency-causing system calls like process forking, file reading, and memory sharing, which together are called "high-control events" in [8]. Given the information of a specific object such as the pid of a process or the $inode$ number of a file, Backtracker identifies the previous objects and system calls that could have potentially affected a target object, and displays chains of events in a dependency graph. In a Backtracker dependency graph, each node A represents an OS-level object, and each edge $A \rightarrow B$ represents that object B is dependent on object A. Moreover, an edge $A \leftrightarrow B$ is used to represent that objects A and B are potentially dependent on each other. As mentioned earlier, the major limitations of Backtracker are the complexity of its results and the inconvenience to use because of its dependency on the availability of "detection points". In its later version [14], the tool can also track dependencies between remote hosts by tracking the logged socket ids.

Although Backtracker does not monitor all kinds of OS-level events, the process, file, and filename objects are among the most important elements in most attacks' prerequisites and consequences. The high-control events monitored by the Backtracker are the essential methods to modify the system attributes. Finally, although OS-level event logging can be disrupted by kernel-level attacks, kernel-level attacks are a lot more difficult and uncom-

mon. Thus, it is reasonable to assume that most attacks will have their corresponding OS-level events logged by Backtracker.

3 Integrating Alert Correlation and OS-Level Dependency Tracking

Our integration is to identify the relevancy between the relationships among IDS alerts and the dependencies among OS-level objects, and then use the OS-level dependencies to verify or discover the relationships among IDS alerts. To identify such relationships, we first look into attacks' OS-level behaviors.

From the operating system's point of view, an attack is a set of OS-level events that access or modify a set of system objects. The OS-level objects and operations corresponding to an attack can be derived from the semantics of the attack. In our model, such semantics consist of two parts: one is the prerequisites and consequences of attacks, and the other is the correspondence between the predicates in attacks prerequisites or consequences and the OS-level objects on the host. With such information, we can identify the OS-level objects corresponding to the attacks on the host. For example, given an attack that exploits a vulnerable service as its prerequisite and yields a shell as its consequence, we can identify the corresponding service process and shell process for this attack.

Accordingly, the OS-level objects corresponding to an attack can be divided into two sets: the *prerequisite object set*, which are the objects derived from the attack's prerequisite, and the *consequence object set*, which are the objects derived from the attack's consequence. These two sets may overlap, because some attacks' consequences may affect their prerequisite objects. By backtracking among the OS-level objects, we can also find dependencies among those objects at the OS level. Though different from the *prepare-for* relation used in alert correlation, such OS-level dependencies can be utilized to verify or discover the *prepare-for* relations among the alerts.

In our framework, we first extract necessary information from the alerts to identify the corresponding OS-level objects. We then verify the dependencies among alerts by using the OS-level dependencies among their corresponding objects, and thus improve the alert correlation based on the causal relationship. Moreover, by identifying the OS-level objects corresponding to the specific evidence indicating possibly missed attacks, and generating a forest of dependent objects by tracking back from these objects, we can improve the performance of existing methods [11, 18] for hypothesizing about possibly missed attacks.

An attack has to have impacts on the local system in order to be observable in the OS-level log. Since some unsuccessful attacks do not have the same impact on system objects as successful ones, our method only guarantees improvement of alert correlation for the alerts of successful attacks, though it may provide positive results for some failed attack attempts.

3.1 Identifying OS-Level Objects Corresponding to Intrusion Alerts

Now we discuss how to find the OS-level objects accessed by the attacks which trigger IDS alerts. We call this process the mapping of IDS alerts to OS-level objects.

We summarize the semantics carried by an alert that can be used to identify the corresponding OS-level objects. Firstly, an IDS alert comes with a timestamp, which indicates when the attack happens. Secondly, given an alert, we have the knowledge about how the attack works and how the system should behave in response to it. For example, given a Snort alert "FTP EXPLOIT wu-ftpd 2.6.0", we know that the corresponding attack exploits a vulnerable wu-ftpd server and forks a root shell. Finally, given local system's configuration, we can identify which OS-level objects correspond to each predicate in attacks' prerequisites and consequences. For example, a predicate "Samba server" may correspond to "/usr/sbin/smbd" process on a given computer. Below we discuss how each type of knowledge is used to map the alerts.

Though the number of logged events and objects is large in system logs, the timestamp of each alert can be used to easily narrow down the potentially relevant system objects. In Backtracker's log, each

OS-level object or event is associated with a time period, which is the lifetime of the object or event. (The original Backtracker toolkit does not provide exact timestamp information. We slightly modified Backtracker's source code and added this functionality.) Given a fixed time period $T = [t_1, t_2]$, an object o can be accessed during T if $o's$ lifetime $[t_s, t_e]$ overlaps with T. Given the timestamp of an alert, we can estimate an approximate time window during which all the relevant OS-level activities occur, and then narrow down the scope of OS-level objects that need to be examined. Such a time window has to be relaxed to accommodate delays in OS-level operations and the clock discrepancy between the IDS sensor and the OS.

Given the name of an alert, we have the corresponding attack's prerequisite and consequence from experts' knowledge. According to [10], the prerequisite of an attack is a logical combination of predicates, and the consequence of an attack is a set of predicates. Each of those predicates is associated with some OS-level objects such as services, processes, and files. Thus, for each predicate in attacks' prerequisites and consequences, we can identify the corresponding file or process on the host computer, and represent them as ($predicate$, $OS\text{-}level\ object$) pairs in the knowledge base. For example, given a pair ($Samba_service(host_IP)$, "$/usr/sbin/smbd$") in the knowledge base, whenever there is predicate of $Samba_service(host_IP)$, we can locate its corresponding process of "$/user/sbin/smbd$". Thus, after identifying the predicates in an attack's prerequisite and consequence, we can identify the OS-level objects corresponding to those predicates on the host computer. Based on whether the corresponding predicate belongs to attack's prerequisite or consequence, those objects can be divided into prerequisite objects and consequence objects.

There are additional constraints for the mapped objects. Firstly, some predicate implies constraints on the properties of its corresponding OS-level objects. For example, the predicate $Root_shell(host_IP)$ implies the privilege of its corresponding OS-level object is root, which is represented by $uid = 0$ for the corresponding object in Backtracker's log. We represent such a constraint along with the object mapping information in the knowledge base in the form of ($predicate$, $OS\text{-}level\ object$, $constraint$). In the above example, we may have a triple ($Root_shell(host_IP)$, $/usr/bin/sh$, $uid = 0$) to indicate that the predicate $Root_shell$ is mapped to a process $/usr/bin/sh$, and the process's uid should be 0.

Secondly, if we focus on successful attacks, we expect to see the prerequisite and consequence of a successful attack at the OS level. In other words, for each predicate p in an attack's prerequisite and consequence, there must be at least one object o in the mapped object set corresponding to p, unless the logging tool does not monitor objects corresponding to such predicates. For example, assume the prerequisite and the consequence of a Samba buffer overflow attack are $Vulnerable_Samba_service(dstIP)$ and $Root_shell\ (dstIP)$, respectively. If this attack is successful, there must be OS-level objects corresponding to these predicates in the object sets. Thus, if such OS-level objects do not exist, the alert must represent a false alert or a failed attack.

Finally, the system activities corresponding to an attack should all be related in Backtracker dependency graph. Moreover, the consequence of an attack should be dependent on the prerequisite of the attack at the OS-level. Thus, for each consequence object corresponding to an attack, it should be a prerequisite object, or dependent on some prerequisite objects of the attack. Thus, we have the dependency constraint: Given alert A and its mapped prerequisite object set P_A and consequence object set C_A, for each consequence object o in C_A, if P_A is not empty, there should exist paths from objects in P_A to o in Backtracker's dependency graph unless o is also in P_A. If such paths do not exist, the corresponding consequence object o should not be associated with the given alert. For example, given an alert with prerequisite object set $\{service_process\}$ and consequence object set $\{shell_process, file_foo\}$, if $service_process$ is not connected to $shell_process$ in the Backtracker dependency graph, the consequence object $shell_process$ should not be included in the consequence object set.

After mapping IDS alerts to OS-level objects

and with OS-level dependency tracking tools, the number of false correlations can be potentially reduced by verifying the dependencies between the corresponding objects. Below we discuss in detail how to identify the OS-level dependencies among alerts with OS-level dependency tracking, and how to use such dependencies to achieve better accuracy in alert correlation.

3.2 OS-Level Dependencies among IDS Alerts

After IDS alerts are mapped to groups of objects within particular time periods in the Backtracker dependency graph, some groups are connected with each other through objects and events in the dependency graph while others are not. It is not difficult to see that an later object is dependent on an earlier object if there exists a path in the dependency graph from the earlier object to the later object. Such paths among these object groups reveal the dependencies between their corresponding alerts. However, such dependencies can be not only malicious attack behaviors but also normal system activities, which makes it different from the prepare-for relations used in alert correlation. To find out the security-relevant dependencies interested by alert correlation, below we discuss in further detail about these dependencies.

As we have mentioned, the OS-level objects corresponding to an alert can be divided into two subsets: the prerequisite object set O_P, which are derived from the attack's prerequisite, and the consequence object set O_C, which are derived from the attack's consequence. As discussed earlier, the consequence objects in O_C should be dependent on the prerequisite objects in O_P. Moreover, two alerts being correlated with each other means the earlier attack's consequence "contributes" to the later attack's prerequisite. Thus, at the OS level, such a prepare-for relation should be reflected by the paths from the earlier attack's consequence object set to the later attack's prerequisite object set (i.e., the later attack's prerequisite objects are dependent on the earlier attack's consequence objects). To distinguish such dependencies from other dependencies, we say alert A is *strongly connected* with alert B if, in the Backtracker dependency graph, there exists a path from one of A's consequence objects to one of B's prerequisite objects. Thus, if alert A prepares for alert B, A should be strongly connected with B.

3.3 Verifying the Dependencies among Correlated IDS Alerts

As discussed in [8], there are several types of attacks that can evade Backtracker. For example, attacks exploiting modified guest kernels and attacks utilizing hidden channels. Such attacks cannot be accommodated by the techniques discussed in this paper. However, other than a few exceptions, Backtracker is capable of tracking OS-level dependencies among most other types of attacks. In other words, in normal cases, if two alerts are not found strongly connected with each other, there should not be prepare-for relations between them.

Given an alert correlation graph, we can map the alerts to OS-level objects and check whether the correlated alerts are strongly connected in OS-level dependency graph. If two correlated alerts are not found strongly connected, the correlation between the two alerts are considered a false correlation. For example, assume $A \rightarrow B$ are a pair of correlated alerts. To verify this correlation, we first map the two alerts into OS-level object sets. If the mappings are successful, there will be corresponding prerequisite and consequence object sets: P_A and C_A of alert A, as well as P_B and C_B of alert B. By tracking back from the objects in P_B with Backtracker, we can verify whether the two alerts are strongly connected. If there does not exist a path between objects in C_A and objects in P_B, we consider the correlation between A and B false.

Note that two alerts being strongly connected in the dependency graph does not guarantee that the earlier one prepares for the other. This is because OS-level dependencies can be operations of benign programs. That being said, being strongly connected on OS-level dependency graph indicates that the involved attacks have higher possibility to be causally related. Thus, we can use this information to discover attacks missed by IDSs, which lead to missing correlations among alerts.

3.4 Facilitating Hypotheses of Missed Attacks

Integrating IDS alert correlation and OS-level event logging can also help in making hypotheses about possibly missed attacks. Several approaches [11, 18] have been proposed in making hypotheses about possibly missed attacks. Given evidence of attacks being missed, these methods search among known attack types of attacks to fill in the gaps between their correlation graphs and the evidence based on attacks' prerequisites and consequences. However, such search processes can be computationally expensive considering the size of the attack type knowledge base and the number of steps that could have been missed.

Integrating IDS alert correlation with OS-level dependency tracking can facilitate hypothesizing of missed attacks. Since the evidence studied as sign of missing attacks can be either IDS alerts or system objects, such evidence can also be mapped to groups of system objects. Assume evidence E is mapped to a set of OS-level objects. By tracking backward from these objects in the OS-level log, these objects can span a forest of system objects connected via various events. For any missed attack, unless it is one of the attacks that can evade the OS-level dependency tracking tool, part of its mapped objects must be in this spanned forest. Using the information of the correspondence between predicates in attacks' prerequisites/consequences and OS-level objects, this forest of objects can be converted to predicates. Then, the searching space for possibly missed attacks can be reduced to the set of attacks related to these predicates. Also, for each hypothesis candidate, we can validate it by trying to map it to OS-level objects. A hypothesis failing to be mapped is considered as invalid.

For example, an attacker attacks a host in the following steps: (1) Attacker successfully launches a buffer overflow attack toward a vulnerable Samba server, which yields a root shell. (2) The attacker deletes the web page files via the shell. Now assume all those activities are missed by the IDS while the file deletion is detected by some file system integrity monitoring tool, which is taken as evidence indicating previous attacks being missed. By tracking back from the deleted file, the file is found dependent on the following objects in the OS-level event log: a smbd process and a shell process forked by the smbd process. Thus, when searching for possibly missed attacks, we can limit the search within attacks related to Samba and shell. Since only Samba is an initial system service, we hypothesize there is a Samba-related attack missed. By trying to map each candidate attack to OS-level objects, we can eliminate the majority of invalid hypotheses. The uncertainty within the remaining results is affected by the knowledge we have about the local system (e.g., the version of Samba) and the attacks (e.g., the number of attack types in the knowledge base).

4 Experimental Results

We have performed a series of experiments to validate the effectiveness of our method. Because our method requires that Backtracker monitor the victim system involved in attacks, we were not able to use the data sets available for IDS evaluation (e.g., DARPA's Grand Challenge Problem (GCP) datasets), which only include either tcpdump of attack traffic or simulated IDS alerts. To facilitate the evaluation, we developed three attack scenarios in our lab, in which an attacker launches a sequence of attacks against a computer monitored by Backtracker and Snort.

Our target machine is a linux server with a modified 2.4.20 kernel to run Backtracker. The Backtracker was slightly modified to add timestamps of system calls to its log. The server is configured to run two vulnerable services: Samba 2.2.8 and icecast 1.3.11. Snort 2.40 [13] was installed on the server to monitor the network traffic as an IDS sensor. To detect more attacks, we used the "Bleeding Snort Rulesets" [2] with Snort. Because both Snort and Backtracker are running on the same computer, clock drifting is not considered in our experiments. We also injected background traffic during the experiments to mimic an operational network. The background traffic was collected on the target machine when it was connected to our campus network, and was manually verified to contain no attacks toward the target machine. We also injected some failed attempts of wu-ftpd buffer overflow at-

tacks into the background traffic.

We only discuss the experimental results on Scenario 1 in detail in the main text. Description of the other two scenarios and additional information about these experiments can be found in the appendix.

4.1 Details of Scenario 1

Our first attack scenario exploits the vulnerable Samba service on the target server. It includes the following 4 steps:

1. We launched 2 remote buffer overflow attacks exploiting the vulnerable Samba server. A remote root shell session was created for each of the attacks.
2. Through one of the root shell sessions, we transmitted a pre-compiled server (daemon) file for the DDoS tool TFN (Tribe Flood Network) to the target host.
3. We then started the TFN server on the target machine through the same root shell.
4. We used the TFN's client program to communicate with the TFN server on the target server, and directed the target to start SYN flood and UDP flood attacks against another computer.

The above attacks took about 5 minutes. During this period, Backtracker logged 81,613 events. Moreover, the Snort sensor raised 26 alerts about these attacks:

- 9 "NETBIOS SMB trans2open buffer overflow attempt" alerts (No. 1–8 and No.10) for the buffer overflow attacks toward the Samba server,
- 15 "DDOS tfn2k icmp possible communication" alerts (No. 12–26) for the control messages sent to the TFN2K server daemon,
- 2 "ATTACK-RESPONSES id check returned root" alerts (No.9 and No.11) for the server's responses to the "id" commands, and
- 2 "ATTACK-RESPONSES id check returned userid" alerts for the server's responses to the "id" commands.

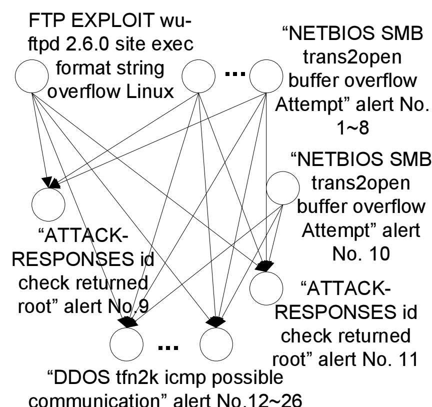


Figure 1. Original Correlation Graph

The background traffic triggered 32 alerts related to the target server:

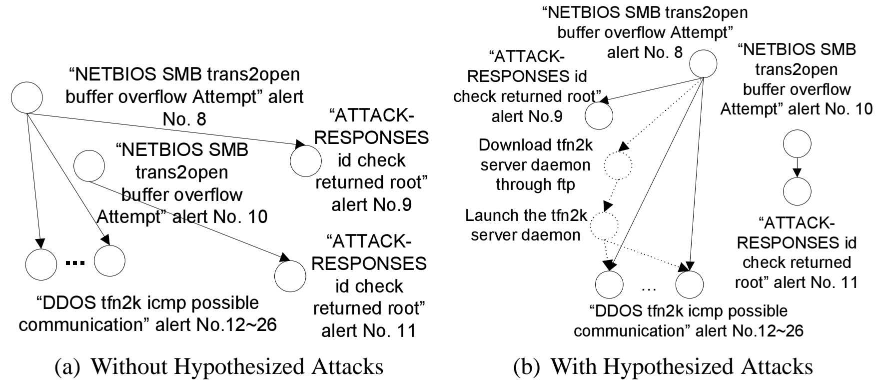
- 8 "SCAN nmap TCP" alerts,
- 23 "SNMP public access udp" alerts, and
- 1 "FTP EXPLOIT wu-ftpd 2.6.0 site exec format string overflow Linux" alert.

Among the above 3 types of alerts, the third one is triggered by the failed attempt of wu-ftpd buffer overflow attack injected into the background traffic.

Using the alert correlation method proposed in [10], we generated the correlation graph shown in Fig. 1. (The prerequisites and consequences of these alerts are given in Appendix B.) Obviously, it contains many false correlations due to the false positives within the reported alerts.

Using the Backtracker's log and the semantics of these alerts, we mapped these alerts to a number of OS-level objects, as listed in Table 1.

For each alert prepared by other alerts in Fig. 1, we generated Backtracker dependency graphs by tracking back from their prerequisite objects. In other words, we want to find in the corresponding Backtracker dependency graph paths from an earlier alerts' consequence objects to an later alert's prerequisite objects if the former prepares for the later. An example of such paths found in our experiments are shown in Fig.3. According to our previous discussion, when there exists such a path, the corresponding alerts are strongly connected and thus the correlations between them are verified at



(a) Without Hypothesized Attacks (b) With Hypothesized Attacks

Figure 2. New Correlation Graphs

Table 1. OS-level Objects Corresponding to the Alerts in Scenario 1

Alert	Prerequisite Objects	Consequence Objects
"NETBIOS SMB trans2open buffer overflow Attempt" No.8	{smbd_2717}	{sh_2720}
"NETBIOS SMB trans2open buffer overflow Attempt" No.10	{smbd_2717}	{sh_2725}
"ATTACK-RESPONSE id check returned root" No. 9	{sh_2722, /usr/bin/id_324551}	Null
"ATTACK-RESPONSE id check returned root" No. 11	{sh_2727, /usr/bin/id_324551}	Null
"DDOS tfn2k icmp possible communication" No.12 26	{td_2737}	Null

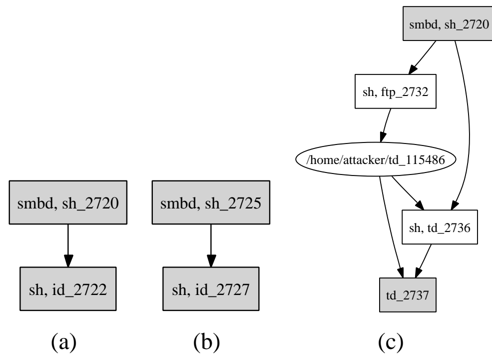


Figure 3. Paths Between Alerts' Corresponding Objects

the OS level. Otherwise, the correlation would be considered false. In this way, we can verify each of the correlations in the original correlation graph, remove those that are verified to be false, and finally come up with a new correlation graph. The new correlation graph for scenario 1 is shown in Fig. 2(a). We can see it is the correct correlation graph of the reported Snort alerts based on the actual attack scenario.

In the correlation graph, the "NETBIOS SMB trans2open buffer overflow attempt" alerts prepare for the "DDOS tfn2k icmp possible communication" alerts, because the consequence of the former (i.e., Root_shell (dstIP)) implies the prerequisite of the later (i.e., tfn2k_server_daemon ()). However, this implication is based on the assumption that an attacker may install the TFN daemon if he/she can create a root shell on a victim computer.

This indicates possible attacks have been missed between these two alerts. By backtracking from the prerequisite object set $\{td_2737\}$ of the "DDOS tfn2k icmp possible communication" alert, a tree of OS-level objects are spanned. Because the consequence object set $\{sh_2720\}$ of "NETBIOS SMB trans2open buffer overflow attempt" alert is among them, we focus on the paths linking the two object sets. Along the path connecting them as shown in Fig. 3(c), we found the following OS-level objects are involved: process object "ftp_2732", file object "$/home/attacker/td_115486$", and process object "td_2736". Thus, instead of guessing all possible ways to install a TFN daemon, which are numerous, we can limit the searching within the activities related to the ftp and tfn2k server program. It is easy to hypothesize that the attacker downloaded the tfn2k server program via ftp and launched it via the shell. These hypotheses are shown in dotted circles and lines in Fig. 2(b).

4.2 Overall Evaluation

Assume an alert correlation method outputs that an alert A prepares for another alert B. If both alerts are detections of actual attacks, and the attack corresponding to alert A is indeed used to prepare for the later attack corresponding to alert B, we consider this correlation as a *true correlation*. Otherwise, it is considered a *false correlation*. Moreover, if one attack is used to prepare for another attack, but there is no correlation corresponding to these attacks (due to missing detection or incorrect correlation), we say there is a *missing correlation*. In our experiments, since we know the details of the attack scenarios, we can easily identify true, false, and missing correlations.

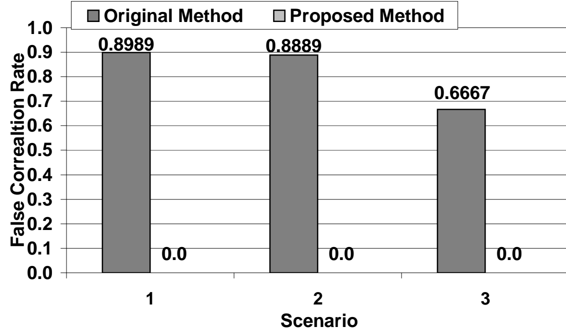
We use two metrics, false correlation rate and missing correlation rate, to evaluate the overall performance of alert correlation before and after integrating Backtracker's results. Given a set of correlated alerts, the *false correlation rate* is the ratio between the number of false correlations over the total number of correlations generated by alert correlation. The *missing correlation rate* is the ratio between the number of missing correlations over the total number of correlations between *attacks*. Intuitively, false correlation rate shows how correct the identified correlations are, while missing correlation rate demonstrates how complete we can identify the correlations between attacks. Obviously, the smaller these two metrics are, the better performance alert correlation has.

Fig. 4(a) shows the false correlation rates in all three attack scenarios. As we can see, our proposed method reduces the false correlation rate significantly in all three scenarios. Indeed, false correlations are completely removed in all scenarios. This is not surprising, because OS-level dependency provides another way to properly verify the correlation between alerts through trustworthy information kept in OS-level logs.

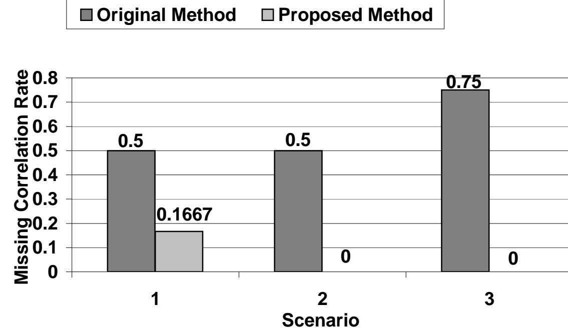
Fig. 4(b) shows the missing correlation rates in the three attack scenarios. We can see significant reduction in missing correlation rate in all three scenarios. While the missing correlation rate is reduced to 0 in scenarios 2 and 3, the missing correlation rate of the first scenario is still non-zero after making hypotheses. This is because the DDoS attack (via the tfn server) is neither detected by Snort, nor hypothesized by our approach.

Our experiments also showed that OS-level object dependency graphs can often be too complicated to understand in reality. For example, during the 30 minutes' attacks in the third attack scenario, Backtracker logged more than $410,000$ events, and the resulting dependency graph contains more than $4,000$ nodes. Analyzing such a complicated graph requires a lot of human experts' time and detailed knowledge about attacks' OS-level behaviors. However, instead of analyzing the Backtracker dependency graph directly, our method uses it as complementary evidence for IDS alert correlation. Because the method *verifies* alert correlations instead of *detects attacks*, only moderate information about attacks' OS-level behaviors is required, and the verification of whether there exists strong connections between correlated alerts can be automatically done by computer programs instead of human experts.

The experiment results also show how the proposed method can help make hypotheses about possibly missed attacks. In the attack scenarios used in our experiments, there is one type of attack that is



(a) False Correlation Rate



(b) Missing Correlation Rate

Figure 4. Compare between the Original Correlation Method [10] and the Proposed Method

hard to detect by all types of IDS, which are the attackers' "legitimate" activities after the break-in. Without the OS-level dependency information provided by Backtracker, it would be quite difficult to guess about such activities that prepare for the later attacks.

5 Related Work

Our technique is closely related to the alert correlation techniques based on prerequisites (also called pre-conditions) and consequences (also called post-conditions) of attacks proposed in [4, 10, 16] as well as OS-level dependency tracking techniques proposed in [8, 14]. These techniques have been discussed in the Introduction and Section 2. The work closest to ours is [14]. In [14], King et al. proposed to use the dependency analysis result from Backtracker to study attacks within and across the hosts, as well as prospected the application of Backtracker in alert correlation. Although they demonstrated the potential benifit of combining backtracker analysis with IDS alert analysis with some heuristic case studies, the main focus of the paper is about backtracking among remote hosts and they did not provide a specific method of combining the two kinds of analyses. The proposed analyses are done manually, which we have pointed out to be potentially very expensive and inpractical. In our approach, after discussed the difference and resemblance between the OS-level dependencies and the dependencies among attacks, we proposed a specific method to automatically combine the analyses after the knowledge base is built.

Several techniques have also used local system information to reason about the causal relationships between IDS alerts. In [18], local system state information is used to reason about the correlation of alerts. In [1], such information is used to analyze the vulnerability of the system and study the potential attacks that could compromise the system. These techniques are complementary to the approach proposed in this paper.

Some approaches have been proposed in [11, 18] to make hypotheses about attacks possibly missed by IDSs. [11] makes hypotheses based on the prerequisite and consequences of attack types when similarities are found between alerts in separate correlation graphs. [18] uses similar attack type knowledge to make hypotheses upon inconsistencies between observed facts and alert correlation graph. Our techniques can facilitate the hypothesis process by bringing additional information from OS-level dependency tracking, thus reducing the search space for the possibly missed attacks.

6 Conclusion and Future work

In this paper, we developed a series of techniques to integrate the alert correlation method (based on prerequisites and consequences of attacks) and OS-level object dependency tracking. A critical step in this integration is to map IDS alerts to OS-level objects, so that connections between alert correlation and OS-level objects can be established. We also identified a number of constraints that the OS-level objects should satisfy if they are relevant to the IDS alerts (or attacks) that are correlated. By us-

ing these constraints, we can verify the IDS alerts as well as the correlation between IDS alerts, and filter out false correlations. Moreover, the dependency between OS-level objects can also facilitate the hypotheses of attacks possibly missed by the IDSs by tracking OS-level objects. Our experimental evaluation gave favorable results, showing that OS-level dependency tracking can significantly reduce false correlations when integrated with the alert correlation method.

The proposed method has several limitations. First, it depends on experts' knowledge about attacks' OS-level behavior. Second, it is not as effective against kernel level attacks, which may corrupt the logs kept by, for example, Backtracker. Finally, to use OS-level information, we have to focus on attacks that have impacts on OS-level objects. Thus, it does not provide any performance guarantee for the correlation of failed attack attempts.

Several issues are worth doing in our future research. First, the current experiments were performed semi-automatically. To provide a useful tool, we will implement the proposed techniques as a fully automated tool. Second, we will investigate more techniques to integrate OS-level dependency tracking with techniques to hypothesize about possibly missed attacks. Finally, we plan to perform more experiments in large scale environments such as the DETER test bed.

References

[1] P. Ammann, D. Wijesekera, and S. Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 217–224, November 2002.

[2] BleedingSnort. Bleedingsnort. `www.bleedingsnort.com`. Accessed on Feb. 15, 2004.

[3] F. Cuppens. Managing alerts in a multi-intrusion detection environment. In *Proceedings of the 17th Annual Computer Security Applications Conference*, December 2001.

[4] F. Cuppens and A. Miege. Alert correlation in a cooperative intrusion detection framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, May 2002.

[5] O. Dain and R. Cunningham. Building scenarios from a heterogeneous alert stream. In *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*, pages 231–235, June 2001.

[6] O. Dain and R. Cunningham. Fusing a heterogeneous alert stream into scenarios. In *Proceedings of the 2001 ACM Workshop on Data Mining for Security Applications*, pages 1–13, Nov. 2001.

[7] H. Debar and A. Wespi. Aggregation and correlation of intrusion-detection alerts. In *Recent Advances in Intrusion Detection*, LNCS 2212, pages 85 – 103, 2001.

[8] S. King and P. Chen. Backtracking intrusions. In *Proceedings of the 2003 Symposium on Operating Systems Principles (SOSP)*, October 2003.

[9] B. Morin, L. Mé, H. Debar, and M. Ducassé. M2D2: A formal data model for IDS alert correlation. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, pages 115–137, 2002.

[10] P. Ning, Y. Cui, and D. S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 245–254, Washington, D.C., November 2002.

[11] P. Ning, D. Xu, C. Healey, and R. St. Amant. Building attack scenarios through integration of complementary alert correlation methods. In *Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS '04)*, pages 97–111, February 2004.

[12] P. Porras, M. Fong, and A. Valdes. A mission-impact-based approach to INFOSEC alarm correlation. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, pages 95–114, 2002.

[13] M. Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 1999 USENIX LISA conference*, 1999.

[14] D. L. S.T. King, Z.M. Mao and P. Chen. Enriching intrusion alerts through multi-host causality. In *Proceedings of the 2005 Network and Distributed System Security Symposium (NDSS)*, February 2005.

[15] S. Staniford, J. Hoagland, and J. McAlerney. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10(1/2):105–136, 2002.

[16] S. Templeton and K. Levitt. A requires/provides model for computer attacks. In *Proceedings of New Security Paradigms Workshop*, pages 31 – 38. ACM Press, September 2000.

[17] A. Valdes and K. Skinner. Probabilistic alert correlation. In *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection (RAID 2001)*, pages 54–68, 2001.

[18] Y. Zhai, P. Ning, P. Iyer, and D. Reeves. Reasoning about complementary intrusion evidence. In *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC '04)*, December 2004.

A Additional Attack Scenarios in the Experiments

A.1 Scenario 2

The attack scenario is as below:

1. Gain root shell from samba trans2open overflow exploit.
2. Download the pre-compiled iroffer to the target server.
3. Launch the iroffer program and turn the target server into part of the attack's own P2P file sharing network.
4. Transferring files via this P2P network.

The Snort raised the following alerts toward the attack:

- 8 "NETBIOS SMB trans2open buffer overflow attempt" alerts, and
- 1 "ATTACK-RESPONSES id check returned root" alert, and
- 1 "ATTACK-RESPONSES id check returned userid" alert, and
- 1 "BLEEDING-EDGE P2P iroffer IRC Bot offered files advertisement" alert.

Using the same background traffic, and the original correlation graph is as shown in Fig.5(a). Similar to the analysis in Scenario 1, by mapping the alerts to the Backtracker log and verifying the dependencies between object $iroffer(pid = 3057)$ and $sh(pid = 2848)$, we have the new correlation graph as shown in Fig.5(b). The graph with hypotheses are shown in Fig.5(c), within which the hypothesized attacks and correlations are represented in dotted line.

A.2 Scenario 3

The attack scenario is as below:

1. Gain root shell by exploiting the buffer overflow vulnerability in icecast 1.3.11. (With 1 failed attempt.)
2. Through the root shell, add a new user.
3. Change the user group to root.
4. SSH to the target server using the newly generated user account.
5. Download the gzipped source code of iroffer to the server.
6. Extract the source code and compile it.
7. Launch the iroffer service.
8. Transfer files from the target via the iroffer service.

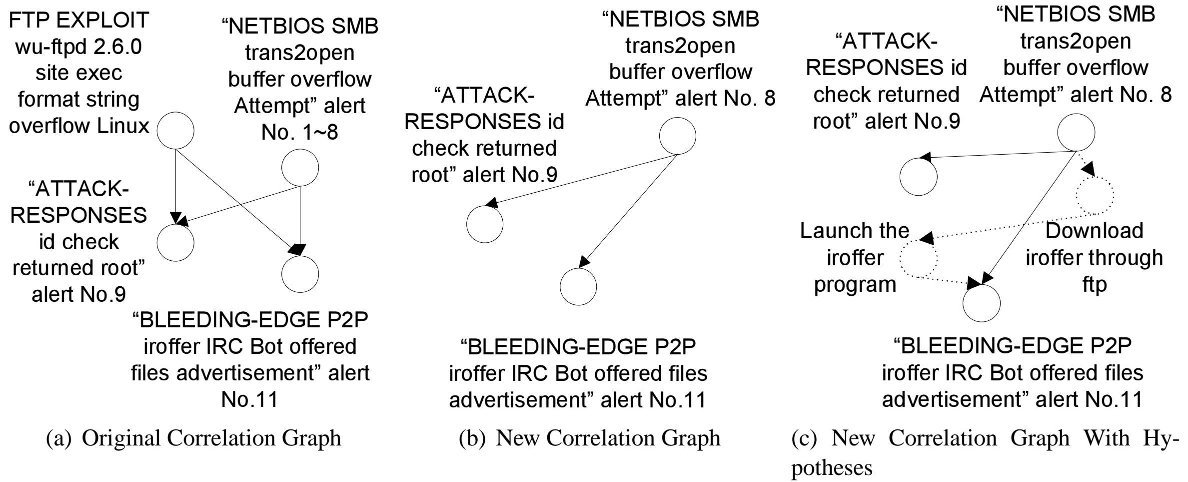
The Snort raised the following alerts toward the attack:

- 2 "SHELLCODE x86 NOOP" alerts, and
- 1 "ATTACK-RESPONSES id check returned root" alert, and
- 1 "ATTACK-RESPONSES id check returned userid" alert, and
- 3 "BLEEDING-EDGE P2P iroffer IRC Bot offered files advertisement" alert.

The original correlation graph is as shown in Fig.6(a). Similar to the analysis in Scenario 1, we mapped the alerts to Backtracker's log. By verifying the dependencies between the shellcode alert ($icecast(pid = 2983) \rightarrow sh(pid = 2996)$ and iroffer alert $iroffer(pid = 11671) \rightarrow socket : /5143$, we have the new correlation graph as shown in Fig.6(b). The graph with hypotheses are shown in Fig.6(c), within which the hypothesized attacks and correlations are represented in dotted line.

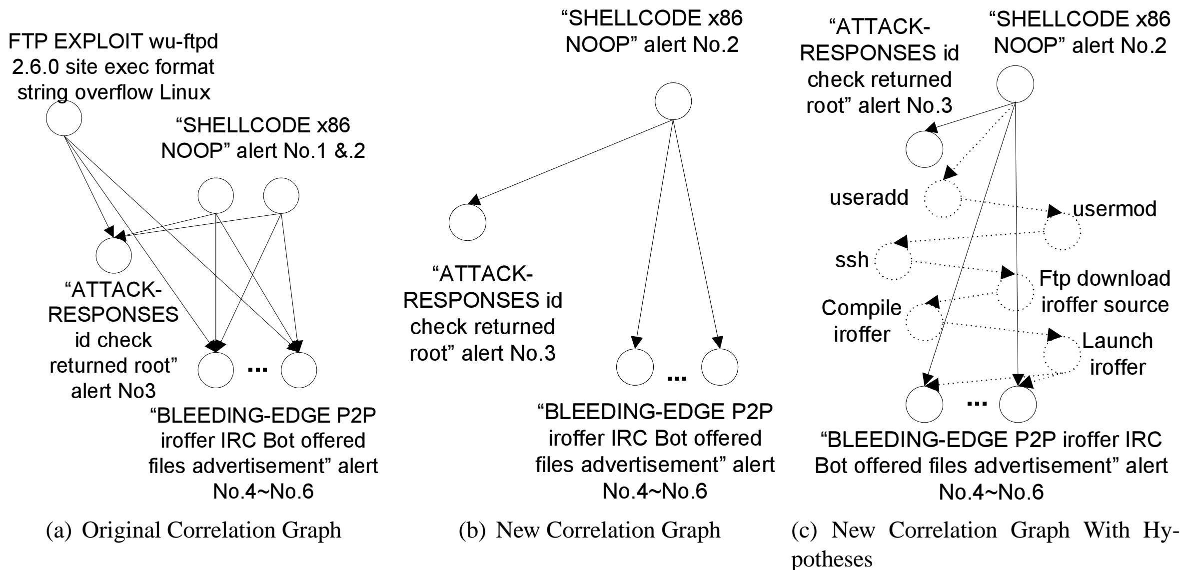
B Attack Knowledge Used in Our Experiments

Table 2 gives the prerequisite and consequence of the attacks in our experiments. Table 3 shows the object mappings for the predicates involved in the experiments. Table 4 presents the implication between predicates related to our experiments.



(a) Original Correlation Graph (b) New Correlation Graph (c) New Correlation Graph With Hypotheses

Figure 5. Correlation Graphs in Scenario 2



(a) Original Correlation Graph (b) New Correlation Graph (c) New Correlation Graph With Hypotheses

Figure 6. Correlation Graphs in Scenario 3

Table 2. Prerequisites and Consequences of Alerts (i.e., Detected Attacks) in Our Experiments. (All alerts have two attributes: Source IP Address (srcIP) and Destination IP Address (dstIP).)

Attack Name	Prerequisite	Consequence
NETBIOS SMB trans2open buffer overflow Attempt	Vulnerable_Samba_service (dstIP)	{Root_shell(dstIP)}
FTP EXPLOIT wu-ftpd 2.6.0 site exec format string overflow linux	Vulnerable_wu-ftpd_service (dstIP)	{Root_shell(dstIP)}
SHELLCODE x86 NOOP	N/A	{Shell(dstIP)}
ATTACK-RESPONSE id check returned root	Root_shell(srcIP) $\wedge$ id_command(srcIP)	N/A
P2P iroffer IRC Bot offered files advertisement	iroffer_service(srcIP)	N/A
DDOS tfn2k icmp possible communication	{tfn2k_server_daemon (dstIP)}	tfn2k_functions (dstIP)
SCAN nmap TCP	N/A	{Gain_host_info(dstIP)}
SNMP public access udp	N/A	{Gain_host_info(dstIP)}

Table 3. OS-Level Objects Corresponding to the Predicates Appeared in Our Experiments

Predicate	OS-level Object	Constraint
Vulnerable_Samba_service (hostIP)	"/usr/sbin/smbd"	N/A
Root_shell(hostIP)	"/usr/bin/sh"	uid=0
Root_shell(hostIP)	"/usr/bin/bash"	uid=0
Shell(hostIP)	"/usr/bin/sh"	N/A
Shell(hostIP)	"/usr/bin/bash"	N/A
id_command(hostIP)	"/usr/bin/id"	N/A
iroffer_service(hostIP)	"/home/attacker/iroffer"	N/A
tfn2k_server_daemon	"/home/attacker/td"	N/A

Table 4. Implications between Predicates

Predicate	Implied Predicates
Root_shell(hostIP)	{Shell(hostIP), tfn2k_server_daemon(hostIP)}