Ntop: a Lightweight Open-Source Network IDS

Luca Deri[1]
Finsiel S.p.A.
Centro Serra, University of Pisa

Almost every new company that connects to the Internet installs a firewall to protect company assets hence prevent unauthorised access to its private network. Despite this general goal, many network administrators decided to improve network security by adding a network-based intrusion detection system (NIDS) able to detect potentially hostile traffic. Although commercial NIDS are quite powerful, they are expensive, often run a dedicated node, and are complex to both configure and deploy.
This paper presents ntop, an open-source traffic monitoring application that features a non-intrusive lightweighted NIDS designed to address the above issues and that can be smoothly integrated into an existing management and security network architecture.

Keywords: Intrusion Detection, Network Monitoring and Security, Internet Technologies.

1. Taxonomy of Intrusion Detection Systems

Network managers realised that a firewall is not enough for protecting a network [12]. This is because:

¥ The firewall might not be properly configured to stop all the suspicious traffic.

¥ Attacks do not always originate on the outside network.

¥ Most hackers do not expect creative defences so they assume that once they are in, nobody is watching their nasty activities.

¥ Firewalls cannot be completely trusted as new attacks appear every day so it is a good practice to add further controls.

As stated above, network managers realised that they cannot rest comfortably knowing that their network is defended only by a firewall, because firewalls represent only one perimeter on a corporate network. For these reasons IDS are rapidly becoming a valuable resource for which many network managers are willing to spend a significant slice of their budget.

Basically, there are two main IDS paradigms [17, 11, 8]:

¥ Anomaly Detection IDS (AD-IDS).

¥ Misuse Detection IDS (MD-IDS).

An AD-IDS [7] needs to learn how the network where it is installed works: it builds some statistical models updated over the time, that describe the topology, the available network services and host traffic changes over the time. As new traffic does not match the model, depending how the traffic differs from the model, it is labelled as suspicious or bad and an alarm is emitted. The main limitation is that if the IDS is too conservative, then it is not really useful because it will generate many false positives (a false alarm) hence on the long term the administrators will ignore it. On the other hand, if it rapidly adapts the model as traffic condition change, it cannot detect much hence it cannot help network administrators. This limitation is due to difficult network traffic modelling as the network increases and becomes more heterogeneous.
A MD-IDS [21] has a database of known attacks and constantly tries to match the actual traffic against the database. If there is a match, an alarm is generated. The database usually contain rules for:

¥ Protocol Stack Verification
RFCs (Request For Comments) specify how the IP protocol stack should work. Attacks often exploit some IP weakness -often due to some incomplete RFC specification- or stack imple-

[1]. Finsiel S.p.A., Via Matteucci 34/B, 56124 Pisa. Centro Serra, University of Pisa, Lungarno Pacinotti 43, Pisa, Italy. Email deri@ntop.org, http://www.ntop.org/.

mentations flaws. Known attacks include the Ping of Death, stealth scanning, and improper use of the TCP three way handshaking. It is worth to note that broken hardware can also generate invalid traffic, not often filtered at source (e.g. on the hub), that could appear as an ongoing attack.

- Application Protocol Verification
  Intruders often exploit application protocol weakness for crashing applications or breaking into hosts. Attacks such as WinNuke and invalid packets that cause DNS cache corruption fall into this category.
- Application Misuse
  Due to some implementation bugs, it is possible to cause applications to crash or gain superuser privileges by sending some specific strings. These problems are often due to buffer overflows.
- Intruders Detection
  Known attacks can be recognised by the effects caused by the attack itself. In particular, intruders usually install daemons or make some traffic (e.g. BO2K default port is 31337) that significantly help network administrator to locate the problem.

Due to the way this paradigm works, it is very unlikely that the system generates false positives. The drawback of a MD-IDS is conceptual. If a network administrator loads the IDS database with a rule for detecting the attack X, then it does not make much sense to have an IDS that warns if the attack X occurred somewhere in the monitored network. This is unless the administrator has properly instrumented the firewall/router to block X or if the IDS is used to check the firewall activities. In fact, due to the static nature of the IDS (the IDS can detect only known patterns), the network will still be vulnerable to new attacks not yet known to the IDS.

Seen the limitations of the above paradigms [14], a new IDS paradigm based on a so-called burglar alarm [18] has been developed. Instead of trying to look for the silver bullet, a burglar alarm relies on the understanding of the network and what should not happen with it. Network managers have to generate a set of rules able to model how the monitored network should conceptually work, so that an IDS can raise an alarm when suspicious traffic is detected. This paradigm is very different from the previous ones: instead of building rules for detecting what look for, the system is instructed to alert the administrators when it detects things that should not normally happen in healthy networks. The richer the database of rules, the more attacks are detected.

Ntop features a lightweight NIDS for IP networks that falls mostly in the last category. As a host-based intrusion detection system (IDS) monitors specific user and application actions and log entries, a NIDS monitors traffic on the wire for specific traffic patterns. As defined in [20], a lightweight IDS should be cross-platform, have a small system footprint, and be flexible enough to be used as permanent element of the network security infrastructure without requiring monitoring or administrative maintenance.

Goal of this paper is to present ntop NIDS. The main contribution of this work to the research field is the novel combination of anomaly and misuse detection approaches. The anomaly-based component (the Network Knowledge Database or NKD) uses automatic discovery of network topology and embedded representation of expected traffic patterns. The misuse component matches specific patterns that are not expected to be seen on the network (burglar alarms). Another contribution to the research field is the tight integration of traffic rules with NKD that made possible to define rules independently of the monitored network: write rules once deploy them anywhere.

The following sections show the ntop architecture, the NIDS design goals, some implementation details, a few scenarios where it can be profitably used, and the unique features make it different from other available IDS.

2. Ntop NIDS Design

Similar to the Unix top tool that reports processes CPU usage, the author needed a simple tool able to report the network top users (hence the term ntop) for quickly identifying those hosts that were currently using most of the available network resources. For this reason ntop was originally designed as an open source [19], web-based traffic measurement and monitoring application, easy to deploy by network administrators. Ntop architecture is rather simple:

¥ The packet sniffer is responsible for capturing packets from the specified network interfaces.

¥ The packet analyser processes captured packets, correlates packets belonging to the same flow (e.g. TCP connection), learns about the network topology (e.g. maintains a list of the current network routers and DNS servers), maintains traffic statistics of known hosts.

¥ The report engine contains an embedded HTTP/HTTPS server that allows users to view traffic reports using a web browser. The server sports access lists, TCP wrapper support and user authentication for restricting ntop access.

¥ Plugins allows developers to selectively extend ntop without having to modify the ntop core; currently ntop sports some plugins for in depth analysis of protocols such as ARP, ICMP, and NFS, or for accessing ntop via ODBC/SQL/SNMP.

The figure below depicts the ntop architecture:

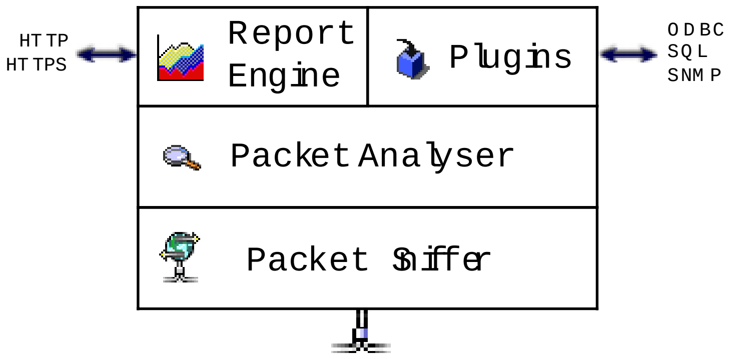


Fig. 1. Ntop Architecture

Although the features described above were satisfactory for many people, some users complained about some ntop limitations:

¥ It provides no means to notify the network administrators about suspicious traffic or obvious dangerous situations rather than through the web interface (i.e. network administrators have to constantly glance through ntop generated pages).

¥ It includes some controls on the monitored traffic but provides network administrators no way to extend such controls without having to modify the source code.

Another open issue is that software that manages and controls a corporate network is made of several interacting components. Many people used ntop in addition to the corporate management platform or, on small organisations, delegated to ntop the control of the whole network. In both cases, ntop was used as a permanent element of the management infrastructure hence it had to interact with other components using standard protocols such as SNMP (Simple Network Management Protocol) [1]. For the above reasons ntop has been extended with a simple scripting language that allows network administrators to specify what network events are considered interesting, and perform some actions (e.g. emit an alarm) as they occur.

So far, ntop NIDS design is not much different from other NIDS. What makes ntop NIDS unique is its knowledge of the monitored network. While capturing packets, ntop learns network topology and hosts relationships (i.e. routers, DNS, networks) and stores this information in a network knowledge database (NKD). This knowledge is dynamic and not specified at ntop start-up by means of configuration files. For instance, if host X successfully routes packets for host Y, then ntop assumes that X is a router for host Y. Similarly, if host K sends packets with different source IP addresses and a single MAC (Media Access Control) address, then K has enabled IP aliasing support. Ntop knowledge database is updated as new packets are captured and is not static whatsoever. As this approach is based on passive network monitoring, the main drawback derives from the limited network knowledge that ntop can have if only few packets have been captured. In addition, as ntop can be activated in a network where there are some misconfigured hosts or where there is an ongoing attack, ntop cannot speculate about the monitored network. For instance if host W sends OSPF (Open Shortest Path First) packets to the router J, then ntop cannot guess that W is a router until it successfully routes packets because W could have installed OSPF just to fool J, hence to manipulate its routing table. For this reason ntop is rather conservative about network topology and learns of it only from the following facts:

¥ ARP packets are used for learning about IP/MAC address association.

¥ IP packets received/sent to non-local addresses are used to identify gateways.

¥ Positive DNS replies identify DNS servers.

¥ Multiple packets with the same MAC address and different IP address identify aliased interfaces.

The reader can point out that intruders can still send forged packets hence fool ntop. This is possible of course, but it is not a major problem because:

¥ Due to the way ntop NIDS rules have been defined, false assumptions about network roles (e.g. a host is wrongly supposed to be a gateway) usually can generate false positives.

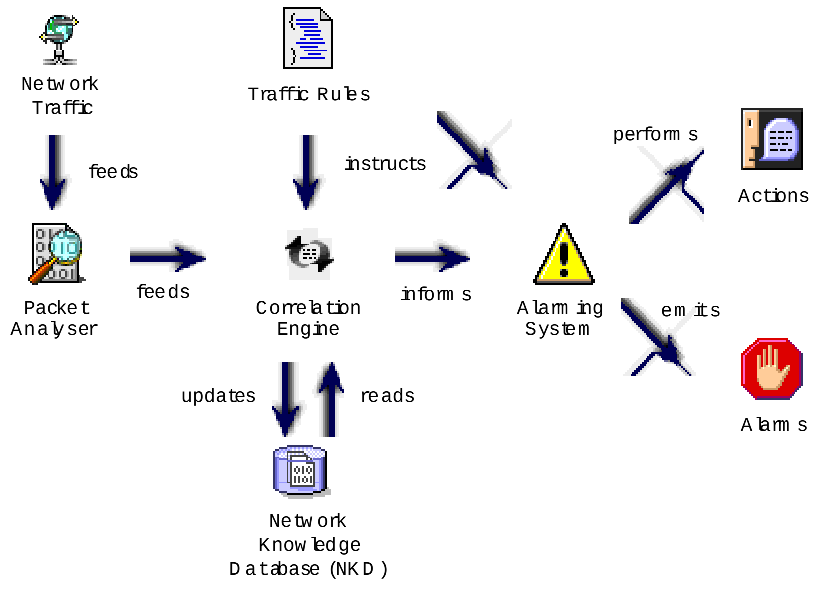
¥ The NKD is constantly updated so forged information is likely to be removed soon and when an inconsistency is found an alarm is generated.

Network knowledge simplifies the task of specifying rules for defining burglar alarms. The following example justifies this statement. In a network, routers are the only hosts entitled to send ICMP (Internet Control Message Protocol) redirect. If host X receives an ICMP redirect from host Y and host Y is not used by X as router, then either Y is misconfigured or Y is trying to capture X traffic by asking X to route its traffic through it. Without dynamic network knowledge, a user the XYZ IDS has to specify the IP addresses of all the known/used routers and then define one rule for each router. Ntop instead, can do the same job using exactly one rule (burglar alarm): `"icmp route-advertisement ICMP_ROUTERADVERT !defaultrouter/any alarm"` that means emit an alarm labelled route-advertisement whenever host X receives an ICMP route advertisement from host Y and Y is not a router used previously by X. It is worth to note that the previous rule does not specify the IP address of X and Y, nor the number of possible hosts/routers. Instead, the rule specifies a fact that will always hold regardless of the IP address, network topology, and media type. As shown in this example, network knowledge exploitation allows an IDS to:

¥ Define fewer rules for a network event (advantage: shorter packet processing time).

¥ Specify rules independently of the network where the IDS is used (advantages: less error prone, network-independent rules, rule portability across networks with both dynamic -by means of DHCP (Dynamic Host Configuration Protocol)- and static addressing).

The network knowledge disadvantage is that a rule is not effective if the knowledge database is small and if an attacker is able to fool ntop. In this case ntop will make some wrong assumptions and then will not be able to recognise specified traffic patterns. Nevertheless, as specified before, ntop is rather conservative about network knowledge hence the risk of fooling ntop should be rather low.



Another important feature of ntop NIDS is the embedded event correlation engine. When a rule is matched by the current traffic an event (either informational, warning, or alarm) is emitted. Usually these events are sent to the management console, if present, or delivered to network administrator by various means including email, and SMS (Short Messaging System). Unfortunately if a rule matches, it is likely that the same rule will match again in the near future because if the problem is not promptly solved then it will happen again. Suppose for instance that host X is attacking host Y with a DOS (Denial of Service) attack. X will probably send a large number of packets with the TCP SYN flag set to the target host. If ntop contains a rule that says if a host sends more that 50 packets in 5 seconds with the SYN flag set to a target host then emit an alarm , the network administrator will probably receive an alarm concerning the attack once every 5 seconds until the attack is over. Besides the fact that the network administrator might be rather annoyed by receiving all these alarms, the IDS will generate itself a large amount of traffic that could eventually turn the IDS into a DOS tool because it will flood the network administrator host with a storm of messages causing as a waterfall effect.
The event correlator can also be used for recognising attacks by monitoring activities that do not complete successfully. For instance suppose that an intruder is trying to learn about the local network before to begin the attack. Very likely, the attacker will scan the ports of the target host or will try some operations that will eventually fail. For instance, the attacker will attempt to connect to non open ports on the target host, or will ping hosts that are either down or not present. As these operations do not succeed, either the attacker will receive an error message or a missing response. In general these are probe events, so they identify problems only if their rate is higher than a specified threshold in a given amount of time. The event correlator can be profitably used for recognising these problems. In fact, the network administrator can define two rules: emit an event of type A whenever you see a TCP packet with the SYN flag set unless this event is cleared before 60 seconds and clear event A if you see a TCP packet with the ACK flag set on the connection that generated event A . The event correlator first sees the event A, and it waits 60 seconds before sending the event to the network administrator. If within 60 seconds the event A is cleared, then the event is deleted. Instead if the timeout expires, the event is finally delivered to the network administrator because it has not been cleared within the expected timeframe. Without the event correlator it would be more difficult -if possible at all-to detect the above attacks. The use of an extern correlator has the disadvantage that both the NIDS and the correlator need to be kept in sync, and that the events emitted by the NIDS can potentially generate a lot of traffic and will likely be filtered out by the correlator.

As explained above, both the network knowledge and the embedded event correlator make ntop NIDS rather different from other similar applications. The following section covers the rule syntax and explains some implementation details. The reader can find in [3] and [4] detailed information about ntop architecture, implementation details, and scenarios where it can be effectively used.

3. Writing Ntop Rules

The author designed NIDS rules according to what the ntop NIDS was supposed to detect. This means that rules are defined using a simple language designed to be expressive enough for describing things to detect but not extremely flexible and rich of statements. This has been done for avoiding users to jeopardise it and create complex rules such as those handled by tools such as P-BEST [9]. Rules are stored in a file specified on the command line (using flag -R) at startup. Ntop parses the rules, checks rule consistence, and stores them in memory on various linked list depending on the rule protocol (e.g. TCP). Ntop tries to match all the suitable rules for each captured packet as a packet can match multiple rules. This means that the packet processing time increases linearly with the number of specified rules.
Ntop natively includes a lightweight IP protocol state machine. Therefore it natively recognises several violations of the IP RFCs including but not limited to TCP three way handshaking check (e.g. some tools use improperly the TCP three way handshaking for several purposes including remote OS identification), overlapping fragments detection, and maximum packet size (packets longer than the MTU). Using rules, administrator can both enable these native checks and specify traffic patterns of interest.

In the current ntop implementation, each rule (for more information see BNF Rule Language Syntax on page 12) has the following format `protocol rule-label rule-parameters`, where:

¥ `protocol`, can be tcp, udp, or icmp. Support for other application protocols including DNS, HTTP and FTP is under development.

¥ `rule-label` is a unique (among the specified rules) word that uniquely identifies the rule.

¥ `rule-parameters` vary according to the specified protocols. Options order is very important. Supported options are:

¥ `revert`
This keyword is used to specify that the rule matches with shost/sport and dhost/dport arguments reverted. For instance if a rule matched an ICMP ECHO requests and this rule is used to catch ICMP ECHO reply, then the revert keyword has to be used.

¥ `shost/sport dhost/dport`
They specify where the matching packet is originating/destined. In case of ICMP packets only shost/dhost is used. Possible values for shost and dhost are: `any` (any host), `broadcast` (a broadcast address), `multicast` (a multicast address), `gateway` (a host that ntop has identified as gateway because it has been used by other hosts for routing packets), `dns` (a host that ntop has identified as a DNS server because it has been used by other hosts for address resolution), `local` (a host that belongs to the local/specified subnet with respect to the host where ntop runs), `remote` (a host that is not local), `anylocal` (any local host), and `anyremote` (any remote host). The difference between local/remote and anylocal/anyremote, is the following: the rule matches only if several packets match (see below the `pktcount` keyword), using `local` it implies that the rule has to match using the same `local` host, whereas with `anylocal` any local host can match.
Possible values for sport and dport are:`any` (any port), `usedport` (a port used by the host, since ntop startup, for receiving/sending data), a string (service name, e.g. telnet) or the port number (e.g. 23). This is the place where ntop network knowledge is used by the NIDS.

¥ `flags`
SRT identifies IP source routing while TCP header flags are specified as S (SYN), F (FIN), and so on.

¥ `ICMP type`
In the case of ICMP protocol, it defines the type of ICMP packet that will match the rule.

¥`type packet/fragment`
This statement is used to restrict the match only to packets or fragments.

¥`pktsize/pktcount operator value`
This statement is used to further restrict the match. Either pktsize (the size of the current packet) or pktcount (the number of packets that matched this rule so far) can be specified. The operator can be <, >, or =, whereas value is an integer.

¥`contains regular-expression`
Previous statements restrict the match on packet header. This statement is used to restrict the match only to those packets whose payload matches the expression. The code that handles regular expressions is POSIX compliant and it has been borrowed from FSF (Free Software Foundation). The current ntop implementation does not perform TCP stream reassembly hence the match is performed on individual packets. Nevertheless this should not be a problem as the expression to search is often located in the first packet (e.g. HTTP header) or at packet boundary (e.g. FTP protocol).

¥`unit seconds`
This statement is used to specify the amount of time during which the previous match `pktcount` should occur. In other words, it specifies the amount of time in which ntop should receive the above specified number of packets matching this rule. For instance `pktcount > 30 unit 10` means that ntop must capture at least 30 packets that match this rule within 10 seconds since the first match.

¥`action`
An action to execute in case of packet match. Valid actions are:

> ¥`alarm`
> Used to send out an alarm.
>
> ¥`mark`
> Used to mark this packet for future processing. In this case ntop stores information about this partial match (matching rule, hosts, and ports) for future use. Generally the `mark` keyword is associated with the `cleans` keyword (see below) that allows to clean previous matches when some conditions happen or to emit an alarm if such condition does not happen within the expected timeout.

¥`cleans rule-name [all]`
The cleans keyword is used with mark (see above) for cleaning out packets marked using the rule rule-name. A packet cleans exactly a marked packet (if any). In case a packet has to clean all the marked packets using rule-name, the `all` keywords can be specified after the rule-name. Please note that `cleans` works in FIFO (First In First Out) mode. Suppose that a host has matched several times the rule X. In this case ntop will keep a list of X matches for the given host. The `cleans` keyword will clean just the first match of X leaving unchanged the other matches unless the `all` keyword is used.

¥`rearm seconds`
When a rule is matched, it might be necessary to specify that the rule is disable for the current shost/sport/dhost/dport combination for the specified amount of time. This feature is useful for limiting the number of matches within a specified timeframe hence to avoid issuing multiple alarms for a given event (e.g. TCP connection reset).

Using the above rules it is relatively simple to define alarms such as:

¥ `icmp route-advertisement ICMP_REDIRECT !gateway/any action alarm`
Emit an alarm when a hosts receive an ICMP redirect from a host that is not one of its gateways.

¥ `tcp root-ftp any/ftp any/any contains "230 User root logged in." action alarm`
Emit an alarm when somebody connected as root on an FTP server.

¥ `udp new-port-open any/any any/!usedport action alarm`
Notify the network administrator when a host received udp traffic on a port on which ntop has observed no traffic before. Please note that the `usedport` keyword has to be properly used, as this condition will not hold twice.

¥ `tcp syn-without-ack any/any any/any flags S action mark expires 6`
`tcp ack revert any/any any/any flags A clears syn-without-ack all`
Emit an alarm for each TCP connection that has not completed the three way handshake within 6 seconds.

Network knowledge allows users to express in one rule complex facts. For instance, the smurf attack is characterised by ICMP echo requests sent to a broadcast address. The rule `icmp smurf ICMP_ECHO any/broadcast action alarm` can be used to identify smurf. As you can note, the rule does not specify the network address nor its subnet mask. Ntop knows what is the local network hence the user can omit it, making the rule usable in every network without having to adapt it. Another example is ping: if a local host pings another local host several time a hour, is not necessarily a problem because the source host can run a network monitoring application. Instead, if a remote host perform the very same action, then this could be a suspicious action as somebody could monitor local activities and spy the local host. The rule `icmp remote-spy ICMP_ECHO remote/anylocal pktcount > 100 unit 3600 action alarm` says that if a remote host sends more that 100 ICMP echo requests to a local host within an hour an alarm has to be emitted. Please note that the above rule is very different from `icmp remote-spy ICMP_ECHO remote/local pktcount > 100 unit 3600 action alarm`. In the first case a match will occur if a remote host will ping for more that 100 times in an hour a host of the local subnet, no matter what is the target host. In the second case instead, the target host must always be the same.

4. Inside Ntop NIDS

Internally ntop arranges information in a dynamically growable hash table. The hash size is given by the number of hosts currently known by ntop in order to use only the necessary memory and because it is not possible to predict who many hosts ntop will know. The more hosts are learnt the larger is the hash as each host occupies a hash bucket. Each hash bucket identifies exactly one host although a host can support IP aliasing hence it can happen that multiple IP addresses correspond to the same bucket. The bucket size is dynamic (1-3 Kbytes) depending on the traffic that a host performs such as the number of active TCP connections. Initially the hash is rather small (32 entries) and its size increase as the number of known hosts increases up to a specified limit. In general the hash size ranges from a few hundred to a few thousand entries depending on the amount of captured traffic. In order to avoid having a huge hash that contains entries of hosts that have not transmitted for a while, ntop periodically runs a garbage collector whose goal is to remove such entries. Entries are purged according to the time they last sent/received traffic. As the garbage collector can remove many entries, if the hash is too large for the current number of entries, the hash is shrinked in order to release some memory. As entries are purged from memory they are stored in a persistent database based on GNU gdbm with entries indexed by address: MAC address if the host is local, IP address otherwise. Database entries contain all the information that was kept in memory (e.g. host address and traffic statistics) but dynamic information such as the status of active TCP connections. This is done to simplify the database design and also because an idle host is supposed not to have active connections. If ntop captures traffic for an host that is not in memory, it checks whether there is an host entry in the database. If such entry exists ntop resurrects it from the database, otherwise a new entry is created. The use of the database allows ntop to save memory while preserving existing traffic statistics. Ntop handles all the host references (e.g. an established TCP session contains a reference to both session peers) as 16 bit hash indexes and not as 32/64 bit pointers. This allows ntop to save some memory and simplifies ntop debugging as because it is difficult to decide whether a pointer is valid when its address is given, whether is rather simple to make the same decision for a hash index. Everytime the hash size changes the hash entries can change bucket index (this might happen due to the nature of hash tables). Due to the rehash, ntop has to update all the hash references in order to reflect this change and avoid keeping indexes that point to buckets no longer in use or that now contain information related to another host.

Depending on the rule definition, as the packet arrives it is not possible to decide immediately if a match happened. As shown in the previous section, a rule can specify a match if ntop observes within a specified timeout (`unit` keyword) a certain number of packets that satisfy the rule. Ntop stores information about partial matches in several linked lists, one per rule. Partial match information includes source/destination host/port, time of the first/last match and number of matches. If a rule is matched within the specified timeframe the rule action is executed and the host/rule entry is purged. Instead, if the match does not occur, the host/rule entry is purged by the garbage collector (please note that this garbage collector has nothing to do with the one described above and used for purging hash entries). As rules can be temporarily disabled (`rearm` keyword), in case of match memory is not freed immediately. The matching entry is then purged later by the garbage collector or recycled if a new match for the same rule/host happens after the specified timeout. Ntop does not try to match packets against temporarily disabled rules. It is worth to note that a temporarily disabled rule is not disabled for all the hosts but just for the hosts that matched it.
Although the ntop NIDS could appear as monolithic code, the detection engine and the event correlator are separated. The detection engine checks whether the processed packet matches against all the specified rules (it is obvious that the received packet is matched only against rules of the same protocol, e.g. a ICMP packet is not matched against UDP rules). For each match, if any, the engine emits an event that is sent to the event correlator. The event correlator is then responsible for making the final decision about a packet match and correlating multiple events (`unit`, and `cleans` keywords). Emitted events can either be logged in a file, delivered to the network administrators via email or SMS, or sent to the management system via SNMP. In a future ntop release, the alerting subsystem will be extended so that alarms can:

¥ be logged in a file, or in the syslog;

¥ have a script associated that is executed when the alarm is emitted (e.g. the script could autoreconfigure the network so the attack is blocked);

¥ start *tcpdump* [6] in background so that future traffic that would trigger the alarm again is filtered and saved on disk for future, in-depth analysis.

In addition, as actions associated to alarms could potentially generate destructive actions (e.g. network reconfiguration should be done only in extreme cases), a rule should have a risk factor associated, so that the action is performed only when the risk factor is above a specified threshold.

Ntop efficiency is greatly influenced by the number of defined rules and their complexity. Checks on packet header are much quicker and less CPU greedy than searching patterns in the packet payload. However, rule processing time is rather efficient as the rule syntax does not allow users to specify complex expressions or ad hoc rule code. This is because ntop lightweight NIDS design goals are performance and simplicity. In fact, complex rule specification would have affected significantly rule processing time that would have turned ntop in a full fledged NIDS that is outside the scope of this work. Ntop goal is to run in background without interfering with user activities while being able to detect network flaws and security issues. In this view ntop NDIS design philosophy is very similar to Snort [20] and SHADOW [13], rather than NFR [16] and NetRanger [2] that are targeted to large institutions that need a powerful IDS and are willing to pay the price of these tools and their steep learning curve. Ntop NIDS has not been uniquely designed for detecting security attacks. In fact, misconfigured computers and faulty applications are also detected by the NIDS. Ntop comes with a set of rules for detecting traffic that should not flow on healthy networks and other rules for detecting security violations and simple attacks. In the author s experience, the probability of having a misconfigured computer/application on a network is much higher than hosting an intruder hence a NIDS should not limit its scope uniquely to security. In addition, as ntop has been designed for simplicity, the author decided to limit its security features and build an application able to report only true facts rather than make assumptions on future traffic and warn the administrator about potential attacks. This is because in our view it is definitively better to have a tool not capable of detecting some traffic patterns rather than constantly warning network administrators about false problems. In the latter case, the tool will be abandoned soon as it requires a constant human intervention.

5. Ntop NIDS Limitations

The main limitation of the ntop NIDS is that it has to be used as a tool that recognises general security violations (e.g. a host that sends ICMP route advertisements although such host is not a gateway). This limitation comes from the ntop philosophy. Ntop has been designed to be general and able to recognise general problems that should never happen on healthy networks and not to give user the chance to recognise the latest exploits. If the reader is looking for a tool that allow very precise rules to be specified (e.g. the source/destination address, the value of each TCP flag or the value of the fragment id) then tools like Snort should be considered.

Ntop NIDS short-term goal is the detection of network problems by specifying rules that model suspicious traffic. However, in some situations this approach is not very effective. For instance, in a network where during the night the average traffic is of a few Kbps, if there is a traffic peak of 2 Mbps for about an hour this could indicate that a host is affected by a DOS (Denial of Service) attack. This traffic peak could also indicate that somebody is making a disk backup. In both cases, ntop rules are not capable to detect this situation and alert the network administrator. One of the planned rule extensions is the ability to express with rules some conditions on global traffic rather than limiting the scope to packet level. Ntop natively collects traffic information and maintains a list of the network traffic, hence the real challenge is to find a simple way of defining conditions that involve various traffic parameters. In fact the real problem is that traffic patterns are quite complex and involve several measurements than need to be correlated efficiently making difficult to define a simple rule language.

Although ntop rules are simple and compact to write, their simplicity can lead to performance problems as users can write simple rules with a high overhead. The current generation of computers can easily run ntop with the default set of rules on an averagely loaded 100 Mbit network without packet drop. Unfortunately, rules not properly written require a lot of processing power and can lead ntop to drop packets even on 10Mbit networks. For instance the following rule `tcp tcp-syn any/any any/any flags S action mark expires 60` causes ntop to remember for 60 seconds each TCP packet with the SYN flag. If ntop is running on a host that accepts several connections per second (e.g. a web server) the ruleset should contain another rule for cleaning the previous rule,otherwise ntop will have to allocate a significant amount of memory and spend a lot of processing time for a dummy job. At the moment ntop contains a syntax rule checker that is not yet capable of detecting problems as the one above.

6. Future Work

Although the correlation engine sports several interesting features, in the current implementation it is not possible to correlate different rules. Sometimes this capability is necessary, hence we are planning to add this extension and allow users to define filters on alarms. For instance, if the network administrator has defined a rule for detecting smurf attacks and the network management console uses broadcast ping for active hosts detection, ntop would emit several false smurf alarms. Adding a filtering capability on the emitted alarm could avoid the problem.

As discussed before, the ntop engine natively recognises some security violations (e.g. portscan) and reports them to the user in a predefined format. It would be useful to give users the chance to decide:

¥ What security violation can produce an alarm.

¥ The alarm format and the associated actions.

In order to implement this, the rule syntax should be extended in order to include the ability to define the actions associated to native alarms.

Ntop has been conceived as a standalone tool. In some large network a single ntop probe is not enough and a cluster of probes (each one with its own set of rules and NKD), each reporting information about a different part of the network, has to be used. So far, ntop lacks of clustering facilities: it would be interesting to define the clustering requirements and implement them as ntop is often deployed by ISP (Internet Service Provider) on large networks.

As ntop is currently used for statistical network traffic analysis, at the University of Pisa we are investigating whether it is possible to statistically build a model of the traffic flowing across a network. The idea is to identify some traffic parameters that can be used to efficiently model the overall traffic. For instance, ntop is able to report when a generic host makes some traffic on a port not previously used. Statistically, if a host has never done traffic for the X protocol (e.g. FTP) and now accepts incoming X connections, it might be that someone has installed an X server without the administrator authorisation. This is the first step towards network change detection, which is the first step in change control and management, a vital part of intrusion detection.

7. Final Remarks

This paper has presented the design and implementation of a lightweight NIDS based on ntop, an open source network monitoring application. Ntop sports novel features not present on other NIDS such as network knowledge and an embedded event correlator. The goal of this work was not to show that the proposed approach is better than others, but to understand its benefits and limitations and demonstrate that ntop NIDS efficiency, rule expressiveness, and ease of use could make it a permanent element of the network security infrastructure.

8. Availability

Ntop for both Unix and Win32 is distributed under the GPL2 licence and can be downloaded free of charge from both the ntop home page (http://www.ntop.org/) and other mirrors on the Internet. Some Unix distributions including but not limited to FreeBSD and Linux, come with ntop preinstalled.

9. Acknowledgments

The author would like to thank all the ntop users and early adopters who deeply influenced the design of the overall architecture with all their comments and suggestions.

10. References

1. J. Case, and others, *Simple Network Management Protocol (SNMP)*, RFC 1157, 1990.
2. Denmac Systems, *Network Based Intrusion Detection: a Review of Technologies,* http://www.denmac.com/, November 1999.
3. L. Deri, and S.Suin, *Practical Network Security: Experiences with ntop*, Proceedings of Terena 2000, May 2000.
4. L. Deri, and S.Suin, *Effective Traffic Measurement using ntop*, IEEE Communications Magazine, May 2000.
5. S. Garfinkel, and G. Spafford, *Practical UNIX Security,* O'Reilly & Associates, 2nd Edition, ISBN 1-56592-148-8, April 1996.

6. V. Jacobson C. Leres, and S. McCanne, *tcpdump*, Lawrence Berkeley National Labs, ftp://ftp.ee.lbl.gov/, 1989
7. H. Javitz, and A. Valdes, *The SRI IDES Statistical Anomaly Detector*, Proceedings of IEEE Symposium on Security and Privacy, May 1991.
8. S. Kumar, *Classification and Detection of Computer Intrusions,* PhD Thesis, Purdue University, August 1995.
9. U. Lindqvist, and P. Porras, *Detecting Computer and Network Misuse Through the Production-Based Expert System Toolset (P-BEST)*, Proceedings of 1999 IEEE Symposium on Security and Privacy, May 1999.
10. S. McCanne, and V. Jacobson, *The BSD Packer Filter: A New Architecture for User-level Packet Capture*, Proceedings of 1993 Winter USENIX Conference, 1993.
11. B. Mukherjee, and others, *Network intrusion detection*, IEEE Network, 8(3), 1994.
12. NetworkICE Inc., *Why Firewalls Are Not Enough*, http://www.networkice.com/Library/firewalls.htm, 2000.
13. S. Northcutt, and others, *SHADOW*, http://www.nswc.navy.mil/ISSEC/CID/, Naval Surface Warfare Center Dahlgren Laboratory, 1998.
14. T. Ptacek, *Problems with Intrusion Detection Systems*, Network Security Solutions ‘98, Las Vegas, 1998.
15. M. Ranum, *An Internet Firewall*, Proceedings of World Conference on Systems Management and Security, 1992.
16. M. Ranum, and others, *Implementing A Generalized Tool For Network Monitoring*, Proceedings of LISA ‘97,San Diego, California,October 1997.
17. M. Ranum, *Intrusion Detection: Challenges and Myths*, http://www.nfr.net/forum/publications/id-myths.html, 1998.
18. M. Ranum, *Burglar Alarms for Detecting Intrusions*, http://www.blackhat.com/html/bh-usa-99/bh3-speakers.html, 1999.
19. E. Raymond, *The Cathedral & the Bazaar*, O’Reilly & Associates, ISBN 1-56592-724-9, 1999.
20. M. Roesch, *Snort - Lightweight Intrusion Detection for Networks*, Proceedings of LISA ‘99, 1999.
21. G. Vigna and R. Kemmerer, *NetSTAT: A Network-based Intrusion Detection Approach*, University of California, 1997.

11.BNF Rule Language Syntax

```
rule := <tcp-rule> | <udp-rule> | <icmp-rule>

tcp-rule :=  tcp <ruleName> [ revert ] <saddr>/<sport> <daddr>/<dport>
             [ flags <flags> ] [ <pktDiscr> ] [ <pktSize> ] action <action>

udp-rule :=  udp <ruleName> [ revert ] <saddr>/<sport> <daddr>/<dport>
             [ <pktDiscr> ] [ <pktSize> ] action <action>

icmp-rule := icmp <ruleName> <icmp-type> [ revert ] <saddr>/<daddr>
             [ <pktDiscr> ] [ <pktSize> ] action <action>

icmp-type := ICMP_ECHOREPLY | ICMP_ECHO | ICMP_UNREACH | ICMP_REDIRECT
             | ICMP_ROUTERADVERT | ICMP_TIMXCEED | ICMP_PARAMPROB
             | ICMP_MASKREPLY | ICMP_MASKREQ | ICMP_INFO_REQUEST
             | ICMP_INFO_REPLY | ICMP_TIMESTAMP | ICMP_TIMESTAMPREPLY
             | ICMP_SOURCE_QUENCH

ruleName := [a-Z0-9]+

flags := S | P | F | A | R

saddr:= [ '!' ] ipAddress

daddr:= [ '!' ] ipAddress

sport := [ '!' ] ipPort

dport := [ '!' ] ipPort

ipAddress := any | gateway | broadcast | multicast | dns | remote | local
             | anyremote | anylocal
```

```
ipPort := any | useport | servicePort

servicePort := named | ftp | telnet .....

pktDiscr := contains <string> | type fragment

pktSize := pktsize <comparison> <pktLen> | pktcount <comparison> <seconds>

comparison := ’<’ | ’>’ | ’=’

string := "[a-Z0-9 ]+"

pktLen := 0 ... 65535

action := mark [e xpires <seconds> ] | alarm [ rearm <seconds> ] | <clear>

clear := clears <ruleName> [ all ]

seconds := 0 ... 65535
```