Fifth International Conference on Information Technology: New Generations

Application Based Meta Tagging of Network Connections

Mark Wallis, Frans Henskens, Michael Hannaford
Distributed Computing Research Group
University of Newcastle
Newcastle, New South Wales, Australia

Abstract

Modern operating systems offer a large array of features in their network subsystems that support fine-grained access control, monitoring and accounting. Such features allow a system administrator to account and filter outgoing network connections based on attributes such as the destination IP address and port number of the connection. With the increase in multi-user systems such as Grid Networks and Shared Web Hosting, the complexity of these tasks has increased. Current operating systems lack the ability to determine the intent of a network connection based on the connection's technical characteristics alone. This paper presents a new mechanism by which applications themselves are given the ability to pass meta information to the network subsystem, allowing it to take advantage of application specific data.

Keywords: Socket, Tagging, Connection, Accounting, Security

1 Introduction

As complexity in the design of network applications grows, it becomes increasingly important that software systems are able to look at network connections in regards to their intent rather than technical characteristics alone. Network operating systems [11] offer a large array of features that can be used to the advantage of application software, but these features are limited in relation to the information they are able to access.

For example, an operating system may offer a bandwidth throttling feature [4] within its network subsystem which a software package may wish to utilise. These features are often activated based on specific technical attributes of the network connection - such as its destination address or its port number.

With the large array of network connections that exist in any modern system it is often hard, if not impossible, to distinguish the varying intents of each network connection from its technical characteristics alone. This restriction means that advanced networking features in the operating system lay under-utilised by software applications due to lack of ability to tailor features for specific cases.

Examples of such restrictions are found in the areas of Shared Web Hosting [6] and Grid Computing [5]. In these areas it is impossible for the operating system to distinguish between outgoing connections from two separate clients sharing the same infrastructure. There is no differentiating factor in the technical characteristics of the connections that allows the distinction to be made. These examples are explored in more detail in Section 2.

This paper introduces the ability for meta-data to be provided by a network-aware application. The host operating system is provided with the meta-data, allowing the application to inform the OS of the intent of the connection as well as its technical characteristics. Once a connection is established the meta-data is retained, and is available to the operating system. The meta-data information remains secure within the system and it is not transmitted on the connection.

Section 2 of the paper describes the problem in more detail, with specific real-world examples. Section 3 provides the design of the proposed solution and section 4 reviews the security aspects. Finally, section 5 provides information on the developed prototype with testing results appearing in section 6.

2 Problem Description

Intent information cannot currently be passed from an application to the operating system because of the data structures used to represent a network connection in the system. A popular method for representing a connection is the Berkeley Socket [13]. This method allows technical information such as local and remote port, local and remote IP and protocol to be specified. Other auxiliary settings are available by utilising optional function calls once the socket has been established, but these options are generally related to specific operating system functionality such as routing and buffer maintenance.

Various real-world examples demonstrate that knowledge of intent would allow the operating system to provide value-add services to software applications. One key example can be found in the area of Grid Computing [1].

2.1 Grid Computing Example

The ability to provide meta-data to the operating system also has benefits for the field of grid computing. For example, multiple companies may at any one time share the use of a single grid computing deployment. Different companies may lease different network links between the nodes in the grid, and also between the grid and external resources required by deployed software. All network requests leaving a node in the grid currently appear to be owned by the grid process itself, rather than any specific company. It is not possible for the operating system network layer to apply policy-based routing [9] to those outgoing connections. Policy-based routing rules can ensure that each company only uses the network link to which they are entitled, dynamically routing the data based on information other than the destination IP address for the connection.

If the grid middle-ware was able to tag outgoing network connections with a unique ID for the specific company requesting the related data, then the operating system would be able to use this information in making policy-based routing decisions to ensure that each company utilises the correct link. The same approach can be applied to traffic shaping if only one external link was available and the companies each had individual service level agreements [12] that dictated how much of the link they were guaranteed for their data.

3 Socket Meta Tagging

A new piece of information called the connection meta-tag has been defined to allow an application to provide the operating system with information on the intent of the network connection. This meta-tag bridges the information gap between an application and the operating system. The meta-tag allows the application to influence the way the network connection is seen by the networking subsystem of the operating system and any other kernel-level processes. The way the operating system interprets a meta-tag is defined by the system administrator using a rules engine. This rules engine defines how the operating system must act when it is transmitting a packet of information attached to a meta-tag enabled socket.

Each application software package defines its own set of meta-tag values. These values are provided to the system administrator as part of the software documentation to allow the administrator to decide if they wish to implement any specific networking functions based on that meta-data. For example, an application may choose to always meta-tag their network connections with a client ID value. Another application may meta-tag their network connections with a piece of data that represents that connection's priority to the system.

Four properties of meta-tagging must be considered - the structure of the meta-data, the application interface for creating the meta-data, the kernel interface for storing the meta-data and a rule-set which can inform the operating system how to handle a connection with a specific piece of meta-data.

3.1 Data structure

The initial design for the meta-data can be broken down into two components. Firstly, a unique Application ID value represents the software package that is providing the meta-data. This Application ID is assigned from a central authority, allowing the operating system to identify the piece of software requesting the network socket. Application IDs are assigned to software packages rather than to software instances. This makes the IDs portable from one system to another. The central authority model is similar to that used whenever an application requires a new official network TCP or UDP port number. Providing an application ID within the meta-data also ensures that no two applications are able to overlap with their tag data.

The second piece of information is the tag itself. The concept of tagging an object is common in many Web 2.0 applications [8]. A tag is a custom piece of information, specific to the host application, that provides meta-data to any application viewing that object. For example, in our web hosting example the tag value may be set to a client ID to allow accounting rules to store data usage against a particular web host client.

The packaging of these two pieces of information into a shared data structure gives flexibility in the makeup of the tag itself. It is beneficial, for example, to support multiple tags per socket. This definition can be implementation specific, but the recommended approach is to include a linked list of tags against each socket.

3.2 Socket interface

The socket interface is used by software applications when they wish to request a new network connection from the operating system. A socket is defined as the endpoint to a network connection. The Berkeley Socket [13] design contains various API function calls, with the two of interest to this work being the *socket()* and *connect()* calls.

The *socket()* call is used to establish the various data structures required for a new network connection. When an application wishes to establish the network connection it calls the *connect()* function, after which data may be sent out of the system. The *socket()* function call must always be executed before the *connect()* function call.

Extending this behaviour to support application meta-tagging involves the introduction of steps between the *socket()* and *connect()* calls so the application can attach a tag to the socket before it is established. These additional function calls take the application ID and tag information from the user-space application, and copies it into the kernel-space data structure used to represent the socket. As a result, the information becomes available to any kernel-level processes that have sufficient security privileges. Security implications of allowing meta-data to be pushed into kernel-space from a user-space application are discussed in section 4. Inserting the new function call between the *socket()* and *connect()* calls ensures that the meta-data information is established before any packets are created. Figure 1 shows the order of operation.

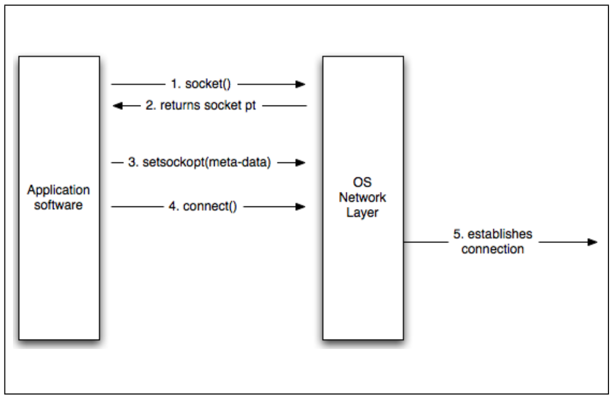

Figure 1. Example of meta-data being applied to a socket

3.3 Rule Set

The kernel-level operating system processes now have the meta-tag information available to them. A rule-set instructs the operating system how to use this new information. Each rule in the rule-set consists of the following components:

1. a condition defined by a set of parameter names and values
2. an action which is performed when the condition is true
3. a focus, which defines when the rule is applied

The condition of a rule can be defined using multiple parameter name/value pairs. The action defines what the operating system should do if it interprets a true response from the condition statement. The focus indicates when the operating system should apply the rule - either on outgoing connections, incoming connections or connections that are being forwarded through the system. To process the rules a rules engine must exist within the operating system. At present, not all operating systems offer a rules engine capable of executing a user-defined rule set, but it is known to be present in Linux [14] and BSD [7]. These rules engines currently exist primarily to act as network firewalls and policy-based routers. The parameters to the conditions in these rule-sets are limited to the standard socket information such as destination and source IP address. The addition of the meta-data on the socket allows this parameter set to be extended to include the tag information as well. The existing rules engines can then be used to trigger actions based on meta-data attached to the network sockets.

4 Security

The application meta-tagging approach allows user-level applications to push custom data into kernel-level data structures. To protect against potential security issues, specifically buffer overflows, the networking function calls that allow an application to set a meta-tag implement a data validation step. Since these function calls are the only way applications can provide meta-data to the socket, they are an appropriate place to implement data and memory security.

Once applications are able to provide additional intent information to the operating system, the issue of trust must be addressed. In particular the operating system must be certain of an application's authenticity before it will act on provided intent information. Without this guarantee it is possible for malicious software to generate false tag information, resulting in, for example, the processing of its network connections at a higher-than-appropriate privilege level. Two possible protection mechanisms against malicious tag generation are *process owner restriction* and *tag signing*.

The simplest protection mechanism involves ensuring that the operating system only trusts tags applied to network connections when the process that requested the connection is being executed by a trusted user, generally *root* or *Administrator*.

The second protection mechanism, tag signing, revolves around the requirement of each software application to digitally sign [2] their tagging request with a protected private key. At the time that an application requests its unique application ID it also can generate a private key for this purpose. The applications private key must be protected from other processes within the system. This method of security allows the operating system to build a trust store of public keys which it can use to verify that tagging requests have come from a valid source. Private keys are paired to the unique application IDs to ensure one application cannot spoof the tags of another. Figure 2 shows the relationship between the components and the security private/public keys. The private key store is protected by the operating system similarly to the encrypted password store used for user authentication.

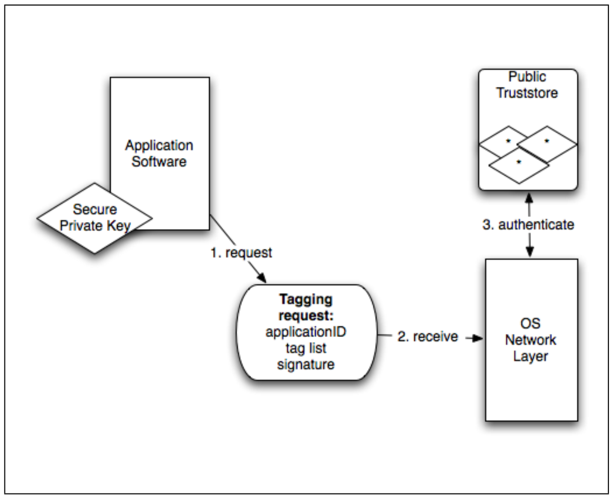

Figure 2. Security implementation using tag signing

5 Prototype Implementation

A prototype implementation of application-based meta-tagging of network connections has been developed. This proof of concept implementation consists of patches and enhancements to the Linux operating system kernel [10] and associated tools. The implementation is broken into four components - a patch to the socket interface, NetFilter modules and enhancements, application code enhancements and an *iptables* rule-set.

5.1 Patch to socket interface

The first component of the example Linux implementation is a patch to the Linux kernel [10] to support the new meta-tag data structure and associated functions.

A new data structure was created in the include/linux/-socket.h header file and defines two integer values used as the meta-data in the prototype implementation. The structure can be tailored to suit specific implementations, for example another application may provide the tag information using a linked list of alpha-numeric strings.

A new socket option titled SO_USERTAG was created to allow the user-space application to attach a meta-tag to the socket . Socket options are set by applications using the *setsockopt()* function call. The *setsockopt()* function call already exists within the Linux kernel and is used for allowing the application to set custom socket parameters such as SO_DONTROUTE and SO_LINGER, which are used to alter the way the Operating System routes the connection and processes its termination.[3].

5.2 Netfilter Module and Iptables extension

An extension to the Netfilter sub-system of the Linux kernel [10] allows the new meta-tag information to be used from the socket data structure. Netfilter [14] is the Linux-based framework that provides the value-add networking services such as policy routing, packet filtering and bandwidth control. It consists of a series of loadable kernel modules, core kernel code and user-space utilities which combine to provide a user-configurable enhanced networking layer inside of the operating system. *iptables* is the user-space configuration tool used to configure the various Netfilter components.

The extension to Netfilter is two-fold:

1. a new Netfilter loadable-kernel-module (LKM) that supports matching on specific meta-tags
2. an extension to the *iptables* tool to allow meta-tag rules to be created

5.3 Example iptables rule-set

With the kernel code enhancements in place the *iptables* command is used to define the meta-tag enabled ruleset. Meta-tags are implemented as an extension to the large pre-existing parameter capability of the Netfilter sub-system. The example command shown in listing 1 defines a rule which requests that the Netfilter code logs to the local syslog whenever a meta-tag with application ID '24' and tag data '26' is encountered.

```
iptables -A OUTPUT -j LOG -m usertag
  --tag 26 --application 24
  --log-prefix "Tagged packet found"
```

Listing 1. Examples iptables command establishing a meta-tag rule

Note that the Netfilter subsystem operates at the packet level rather than the network level, so the rule results in a log entry being written for every packet associated with that network connection. This is the optimal method of implementation as it allows per-packet policy decisions rather than restricting the system to connection-level decisions alone.

The *iptables* command is available for use primarily by the system administrator. Applications are also capable of calling the *iptables* command from their execution code if the appropriate security permissions exist. This allows applications to dynamically create the rule-set on behalf of the system administrator.

5.4 Patch to application code

The final required code change is enhancement of user-level applications which require the tagging capability. Listing 2 presents a snippet of application code in which the meta-tag data structure is built and associated with the socket using the *setsockopt* function. Meta-tagging is completely optional, and this enhancement is only required in applications that wish to define meta-tag data on their outgoing connections. Developers must access the appropriate kernel header files in which the required data structures are defined to compile the required meta-tag support.

When an application defines a meta-tag the application developer must document the tag's definition. This allows the system administrator to build meta-tag rules for that application.

```
struct user_tag_req utr;
bzero((char *) &utr, sizeof(utr));

utr.application_id=1;
utr.tag=1;

setsockopt(sockfd, SOL_SOCKET,
     SO_USERTAG, &utr, sizeof(utr));
```

Listing 2. Application setting a meta-tag on a socket

6 Testing

Various tests were performed on the software prototype to prove the the meta-tagging system was capable of solving the problem as originally stated. Two key aspects of the prototype's performance that were assessed are:

1. Throughput
2. Policy-based routing

6.1 Throughput

During design of the meta-tag system it was imperative that any enhancements made to the kernel networking subsystem would cause no performance penalty on the processing of packets that did not contain a network tag. Speed tests performed using the prototype found no performance degradation was noticeable after an example meta-tag rule-set of 5 rules was applied to the system. It is inevitable that process cycles used by the Netfilter sub-system will affect performance as extra instructions are being processed by the kernel during packet processing, but these effects were so small as to be undetected in these experiments. This performance degradation would also only affect latency of packet transfer, not overall throughput.

Tests were performed using large ($>$ 1 gigabyte) data block transfers over a network socket using the FTP protocol, both with and without the FTP socket being tagged. Table 1 shows speed tests results. While the throughput values themselves are of no particular import, but the fact that they are all very similar shows that the prototype implementation had no noticeable adverse effect on the network performance of the host.

Protocol	*Tagging*	*RuleSet*	*Throughput*
FTP	off	n/a	11.23mbit/s
FTP	on	5	11.14mbit/s
FTP	on	1	11.15mbit/s
HTTP	off	n/a	11.34mbit/s
HTTP	on	5	11.32mbit/s

Table 1. Throughput test results on 1.5 gigabyte data set

6.2 Policy routing

To prove that meta-tagging resolves the grid computing policy routing issue originally described, the rule-set displayed in listing 3 was generated. The initial command establishes an *iptables* rule to match on application ID '1' and tag value '7'. Packets found with this meta-tag have a mark placed on their connection within the kernel networking data structures. The *ip rule* command forces any packets with a mark of value '1' to be routed using the route table labelled *pr1*, rather than the default route table. The *ip route* commands that finish the listing establish different default routes for the default and custom *pr1* route table. This configuration allows packets tagged specifically with application ID '1' and tag value '7' to be routed via a different default gateway to all other packets.

```
iptables -t mangle -m usertag
  --tag 7 --application 1 -j connmark
iptables -t mangle -N connmark
iptables -t mangle -A connmark
  -j MARK --set-mark 1
ip rule add fwmark 1 table pr1
ip route add default
  via 1.1.1.1 table pr1
ip route add default
  via 1.1.1.2
```

Listing 3. Example Policy Routing ruleset

7 Conclusion

This paper presents an extension to operating system and application software to support tracking of the intent of a network connection. Meta-tagging allows operating systems to make sophisticated rule-based decisions on handling of network packets. A prototype of the technique is presented, verifying its feasibility. Specifically, the prototype shows that meta-tagging can satisfy real-world needs. For example the intent of a connection, represented as meta-tag data, can be used by the operating system to provide value-added services such as policy-based routing and accounting to the software application.

The meta-tag information is provided at the connection level. The application of this data, though, is applied on a per-packet basis to allow decisions to be made by the operating system at the packet level.

Of particular interest is the way meta-tagging allows for enhanced services in the area of Grid Computing. Once processes within the grid are able to tag network requests with meta-data, physical network links can be treated as resources within the grid, especially in environments where multiple companies are sharing infrastructure with varying SLAs.

7.1 Future Work

Ongoing research focuses on the following areas:

1. Enhancements to the prototype system to support multiple tags and non-numeric tag values. This is purely an implementation enhancement that will greatly increase the usefulness of the prototype.

2. Investigation into the mapping of tag data to rules and actions. The Linux Netfilter system provides an excellent method of mapping data to rules and actions, using the *iptables* tool, but this feature does not exist in all major operating systems and hence implementation into other operating systems may require the development of such a rules engine.

3. Further investigation into the value-add services this concept can bring to Grid Computing. For example the notion of treating network resources as a reusable resource within the grid itself (just as processing power and data storage are currently treated) warrants further investigation.

4. Dynamic tagging, which would allow applications to remove and add meta-tag information to an established connection.

References

[1] I. Foster. Globus toolkit version 4: Software for service-oriented system. *IFIP International Conference on Network and Parallel Computing*, pages 2 – 13, 2006.

[2] B. Hammond and M. Atreya. *Digital Signatures*. McGraw-Hill/Osborne Media, special co edition edition, May 2002.

[3] Hewlett Packard. *BSD Sockets Interface Programmer's Guide*. Hewlett Packard Press, 1997.

[4] B. Hubert. Linux advanced routing and traffic control howto. *http://lartc.org/howto/*, 2005.

[5] B. Jacob and et al. *Introduction to Grid Computing*. IBM Redbooks, 2005.

[6] D. Kaye. *Strategies for Web Hosting and Managed Services*. John Wiley and Sons, 2002.

[7] S. McCanne and V. Jacobson. The bsd packet filter: A new architecture for user-level packing capture. *USENIX*, 1992.

[8] D. McCormack. *Web 2.0: The Resurgence of the Internet and E-Commerce*. Aspatore Books, 2002.

[9] MIT Laboratory for Computer Science. Rfc 1102 - policy routing in internet protocols. Technical report, Network Working Group, 1989.

[10] C. S. Rodriguez, G. Fischer, and S. Smolski. *The Linux Kernel Primer: A Top-Down Approach for x86 and PowerPC Architectures*. Open Source Software Development Series. Prentice Hall, 2005.

[11] W. Stallings. *Operating Systems: Internals and Design Principles 5/E*. Prentice Hall, 2005.

[12] R. Sturm, W. Morris, and M. Jander. *Foundations of Service Level Management*. Sams Publishing, 2000.

[13] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 4th edition, 2003.

[14] H. Welte. netfilter/iptables project homepage. *http://www.netfilter.org*, 1999-2007.