Toward Robust and Extensible Automatic Protocol Identification

George Louthan[1], **Collin McMillan**[2], **Christopher Johnson**[1], **and John Hale**[1]
[1]Department of Computer Science, University of Tulsa, Tulsa, OK, USA
[2]Computer Science Department, College of William and Mary, Williamsburg, VA, USA

Abstract— *Conventional network protocol identification based on well-known port numbers is no longer sufficient to identify and classify traffic in modern networks. Applications have developed means by which to dynamically change port numbers, users select alternate ports, and attackers attempt to trick identification systems by changing ports. Embracing the position that correctness should not be sacrificed for speed, we introduce an architecture for signature-based, fully stream-aware, automated identification of network protocols, called SAND, which is capable of classifying TCP traffic independently of port number with the goal of obtaining the most demonstrably accurate results possible. The hope is that this work will foster further efforts to revive research into intelligent protocol identification and analysis.*

Keywords: network monitoring, network protocol identification, computer networks

1. Introduction

Network monitoring is a vital aspect of network administration and, notably, enterprise security as well. Clear, reliable knowledge of the state of a network is essential for its effective management. Central to the practice of network monitoring is the identification and subsequent classification of network protocols. Without a reliable means of classifying traffic, even the most advanced appliances cannot block or control it intelligently.

A clear picture of a network is not only useful for security reasons, but also for quality-of-service concerns. In many networks, prioritizing traffic is often highly important, but without a reliable means of classifying that traffic, even the most advanced appliance cannot prioritize or shape it intelligently.

Compounding the problem of traffic classification, users and programs may attempt to evade detection or identification, often to malicious ends. Such attempts commonly exploit the reliance of conventional methods upon well-known port numbers for protocol classification. For example, the peer-to-peer Internet telephony software Skype uses unpredictable port numbers and is capable of remarkably robust NAT and firewall traversal [1]. Though this is a desirable feature from the standpoint of the software's user, it presents a quandary for network administrators seeking to monitor, block, or otherwise manage this traffic.

The state of the art in network monitoring has fallen well behind the needs of modern networks. Because conventional protocol detection no longer suffices to obtain an accurate representation of a network, in this paper we present a solution for the content-based identification of TCP traffic that is fully stream-aware, capable of identifying TCP traffic regardless of port number.

The remainder of this paper is structured as follows. Section 2 presents some background in the field of protcol identification, section 3 describes the architecture and strategy behind our offering, and section 4 provides some analysis of the results of the system. Finally, in section 5 we offer some concluding remarks and describe some avenues for future work.

2. Background

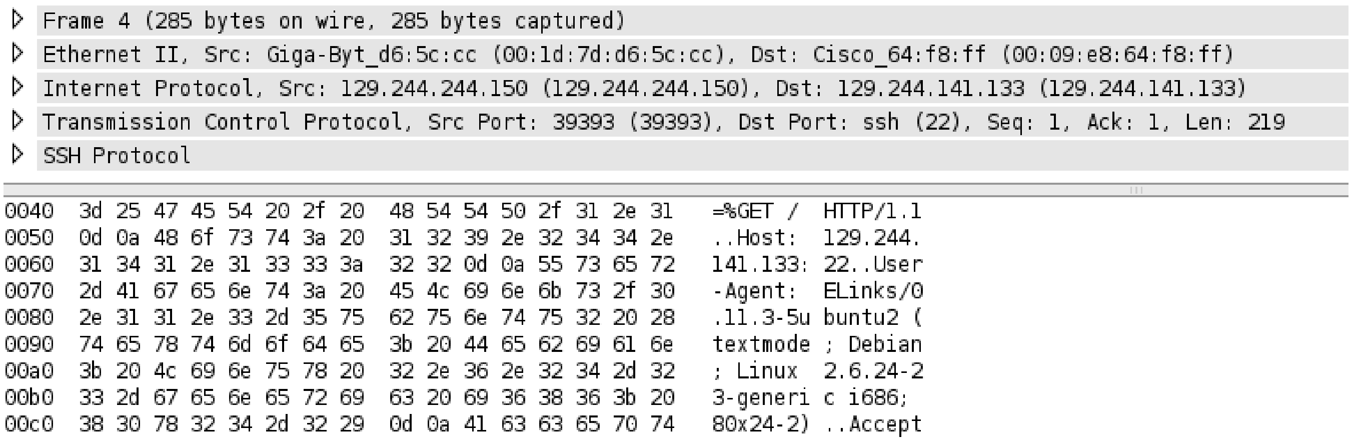
Today, virtually all conventional network monitoring relies upon classifying packets based on TCP or UDP port numbers. This depends on all parties' adherence to well-known or standard port conventions and can result in incorrect or incomplete identification of network traffic.

A typical example of conventional network monitoring is found in Wireshark, a packet sniffer and protocol analyzer. Wireshark performs protocol identification by examining each packet's header and matching the port number with well-known ports for applications and protocols. Once this determination is made, the packet is dissected according to analysis plug-ins applied based upon the port number [2].

Although Wireshark contains sophisticated tools for analyzing and dissecting the payloads of packets, as well as for the analysis of streams, its inability to identify packets based upon the specifications driving those dissectors is a serious drawback. Unfortunately, these protocol analyzers are essentially useless without an initial identification of the protocol.

A demonstration of this failing is illustrated in Fig. 1. In an admittedly contrived (but still valid) example, a web server was configured to bind to port 22, and the traffic between it and a web browser was recorded and analyzed with Wireshark. Wireshark identified the packet as part of an encrypted SSH stream based upon port number. However, to a knowledgeable human observer, the contents of the well-formed HTTP packet would be sufficient for identification.

A new method of traffic classification, capable of reliably identifying application traffic regardless of port number, is needed if network monitoring is to keep pace with its challenges. Furthermore, we believe that for robustness such



```
▷ Frame 4 (285 bytes on wire, 285 bytes captured)
▷ Ethernet II, Src: Giga-Byt_d6:5c:cc (00:1d:7d:d6:5c:cc), Dst: Cisco_64:f8:ff (00:09:e8:64:f8:ff)
▷ Internet Protocol, Src: 129.244.244.150 (129.244.244.150), Dst: 129.244.141.133 (129.244.141.133)
▷ Transmission Control Protocol, Src Port: 39393 (39393), Dst Port: ssh (22), Seq: 1, Ack: 1, Len: 219
▷ SSH Protocol

0040  3d 25 47 45 54 20 2f 20  48 54 54 50 2f 31 2e 31   =%GET /  HTTP/1.1
0050  0d 0a 48 6f 73 74 3a 20  31 32 39 2e 32 34 34 2e   ..Host:  129.244.
0060  31 34 31 2e 31 33 33 3a  32 32 0d 0a 55 73 65 72   141.133: 22..User
0070  2d 41 67 65 6e 74 3a 20  45 4c 69 6e 6b 73 2f 30   -Agent:  ELinks/0
0080  2e 31 31 2e 33 2d 35 75  62 75 6e 74 75 32 20 28   .11.3-5u buntu2 (
0090  74 65 78 74 6d 6f 64 65  3b 20 44 65 62 69 61 6e   textmode ; Debian
00a0  3b 20 4c 69 6e 75 78 20  32 2e 36 2e 32 34 2d 32   ; Linux  2.6.24-2
00b0  33 2d 67 65 6e 65 72 69  63 20 69 36 38 36 3b 20   3-generi c i686;
00c0  38 30 78 32 34 2d 32 29  0d 0a 41 63 63 65 70 74   80x24-2) ..Accept
```

Fig. 1: Wireshark incorrectly identifying HTTP over port 22.

a system must be *content-aware*, making its identifications based upon the actual content of packets that adhere to known, fixed protocol formats. The goal, then, is to leverage in an automated fashion the same data that a human might use to conclude that the example in Fig. 1 is flawed.

Previous work in signature-based protocol identification by Moore and Papagiannaki suggested a 9-layer methodology for identification, but they lacked an automated process for executing their method [5]. Sen, *et al.*, specified an online, automated method for identifying certain specific peer-to-peer protocols. However, their method was not extended beyond peer-to-peer applications, relied upon specialized hardware, and only inspected traffic on a packet-by-packet basis [7].

A free and open source tool called l7filter, a regular-expression based classifier plugin for Netfilter (the Linux packet filtering framework), has achieved a reasonably high level of maturity. However, it effects only rudimentary stream reconstruction by examining, by default, the first 10 packets of a stream, and it is explicitly intended almost exclusively for QoS [4]. It is expected that an attacker could use creative TCP segmentation or other tricks to evade it.

While the body of preceding work has largely sacrificed full TCP stream reassembly in order to obtain higher speeds, making their systems unsuitable as the basis vital security decisions, we have set the specific goal of maximizing the correctness of classification through robust stream reassembly in spite of its overhead. Therefore, in this paper we present a general, automated, self-contained solution for the passive, signature-based identification of TCP-based network protocols using only commodity hardware.

Additionally, although a significant amount of work was done in the field of alternative methods for identifying network traffic several years ago, little has been done since, even though the problem has only gotten more severe. For that reason, a second goal of this paper is to encourage a revival of work in this field and to advocate for the necessity of a new approach to protocol identification.

3. SAND: An Architecture for Signature-based Automatic Network Detection

A solution to the problem of traffic identification needs to fulfill several properties. Speed and extensibility are important, but quality of results are paramount for a system that is to form the basis for making security decisions. That is, it should be fully stream-aware and automated; without satisfying these requirements, an identification system is likely to fall short in robustness and usability, which together are necessary to succeed conventional methods.

Our architecture, dubbed SAND, for Stream-Aware Network-protocol Detector, takes great strides toward exhibiting these properties for detection of TCP-based protocols. The implementation, written largely in Python, provides a module called pysand, suitable for import and use by other applications. The following section describes the general architecture and some implementation details of our detection library.

3.1 Architecture

Fig. 2 shows the basic architecture of the SAND system, which relies upon three basic, modular components: an identification moduSSH is not the only protocol taking this approximate form. For example, in the GNUTELLA protocol, an early version used “GNUTELLA OK” as a server response from a client request. In later versions of the protocol, 0.6 for instance, the response might be “GNUTELLA/0.6 200 OK” [10]. This form, combined with the nature of SAND identifiers, enables the system to deal gracefully with different versions, as well as pass the version (and status code, if desired) back to an analysis module.le (pysand in our reference implementation), which performs the actual protocol detection; one or more analysis modules,

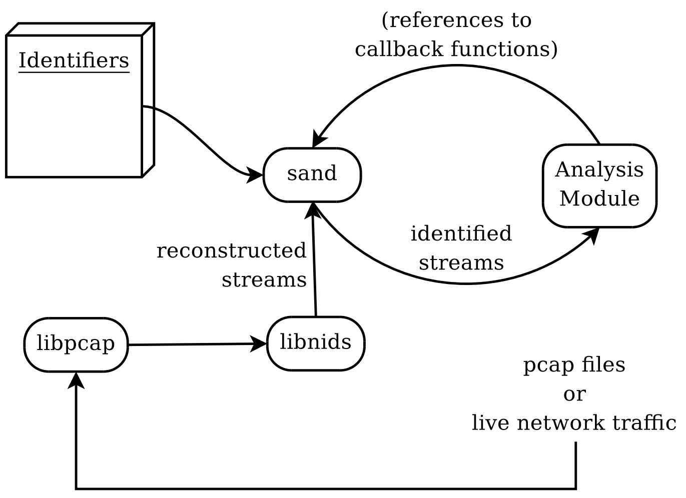


Fig. 2: The SAND architecture.

```
protocol = "SSH"
threshold = 2

[server0]
start = "SSH-"
sig = "s_version"
finish = "\n"

[client0]
start = "SSH-"
sig = "c_version"
finish = "\n"
```

Fig. 3: The identifier for SSH.

which receive protocol information from callback functions they provide to the library; and a set of protocol identifiers, used to specify signatures in order to identify and extract some basic information from the network streams.

Discrete packets are reassembled into a completely correct, consistent stream using libnids, a library providing supporting functions for the design of network intrusion detection systems that actually provides an implementation of the Linux TCP/IP stack and provides the interface to libpcap [8].

3.2 Identification Strategy

The identification strategy depends upon a set of identifiers loaded from files at run-time. A single identifier specifies the information necessary to identify a single protocol and is made up of one or more signatures to be located in the streams.

A single signature in an identifier includes start and end terms, with the unsearched string between them optionally being returned to libsand as potentially useful information. For example, in the SSH protocol, each party must send a protocol identification string of the form "SSH-protoversion-softwareversion SP comments CR LF" [9]; therefore, a sensible identifier for the SSH protocol might find the string "SSH-" and the "CR LF" terminator, taking the string between them to be the protocol version. This identifier is provided in SAND format in Fig. 3.

SSH is not the only protocol taking this approximate form. For example, in the GNUTELLA protocol, an early version used "GNUTELLA OK" as a server response from a client request. In later versions of the protocol, 0.6 for instance, the response might be "GNUTELLA/0.6 200 OK" [10]. This form, combined with the nature of SAND identifiers, enables the system to deal gracefully with different versions, as well as pass the version (and status code, if desired) back to an analysis module.

Like Sen, *et al.*, we noted that protocols often take the form of "stringset1*stringset2" [7]. This identifier format was favored over simpler techniques like plain string matching or string and offset specification, because more power was judged to be useful; and over full-fledged regular expression based string specification system, like what is used in l7filter [4] because the extra complexity was judged unnecessary. Further, the variable content between the two strings is often version information or other desirable content.

Fig. 4 illustrates the general strategy for identifying a stream. Pysand maintains a table of each tracked stream and the protocols they have been matched against, storing a numerical "certainty" value. Each time a signature is matched in the stream, the certainty value is incremented; once it reaches an identifier's specified threshold, stream searching and reassembly cease, and the analysis module is notified of the identification.

4. Analysis

Above, several criteria and requirements are identified against which SAND and other signature-based attempts to solve the protocol identification problem should be evaluated. These included signature-based nature, stream-based processing, extensible architecture, automation, and speed.

SAND identifies protocols according to signatures specified in identifier files. The search system allows the location of arbitrary signatures at arbitrary locations in TCP streams and the parsing of simple information such as version strings based upon specifications made in the identifiers.

While many protocol identification systems, including those is most common use, operate on individual packets (whether on packet headers alone or on their payloads), SAND is fully stream-based. By utilizing an actual implementation of a TCP/IP stack, the system can robustly reassemble packets into streams; this should prove more reliable than any system that depends upon single independent packets.

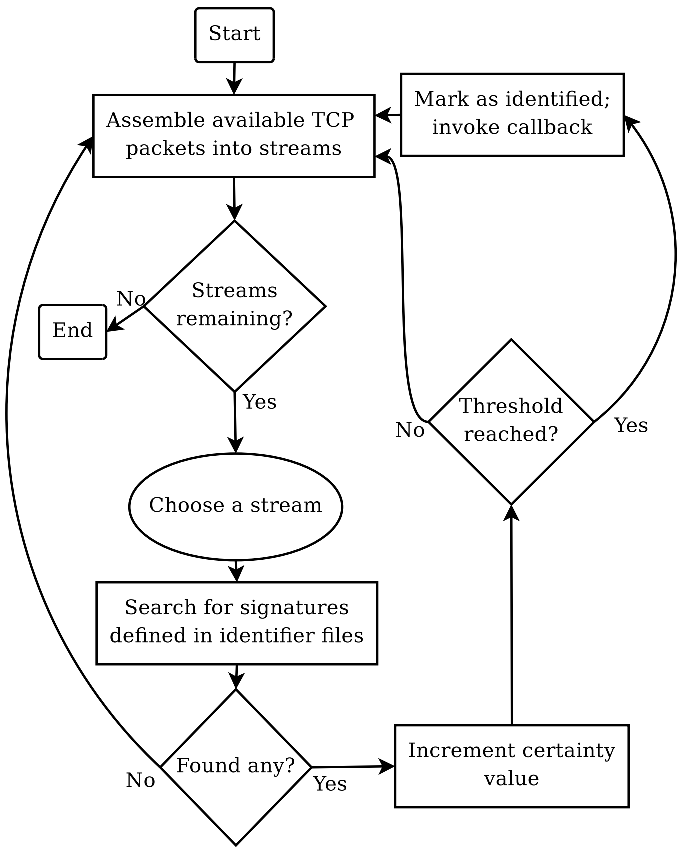

Start
Assemble available TCP packets into streams
Mark as identified; invoke callback
Streams remaining?
No
End
Yes
Threshold reached?
No
Yes
Choose a stream
Search for signatures defined in identifier files
Found any?
No
Yes
Increment certainty value

Fig. 4: The SAND identification strategy.

As an example, Sen, *et al.*, specifically chose to avoid stream reassembly, asserting that the speed gained by eschewing reassembly outweighed potential gains in robustness [7]. However, most identifiable streams can be identified in the first few packets, seriously limiting and, in some cases, altogether eliminating the amount of extraneous reassembly required.

For example, an attacker seeking to avoid detection by a packet-based system could use TCP segmentation to construct packets that do not contain signatures but still deliver equivalent information by altering the size of each packet to ensure that the packet boundary falls in the middle of a signature element, and possibly reordering the packets. The stream reassembly component of SAND ensures that such efforts will fail.

Several other researchers have presented methods of identifying traffic using signatures without providing a clear method for updating, specifying, and extending their systems' ability to recognize different streams from the perspective of a user [5] [12] [7]. In the SAND implementation, a simple, well-defined data file added to the identifier directory introduces a new identifier.

A minor roadblock to extensibility is the requirement for the manual generation of network protocol identifiers. Most common protocols are well studied and well specified, leading to straightforward design of signatures. For other protocols, work has been done to alleviate this problem. Park, *et al.*, created an algorithm called LASER, which seeks to automatically generate application signatures [6]. Although SAND's approach to protocol identification is fully automated, the system itself does not provide any useful method for automatically generating identifiers.

This flows naturally into the question of automation. SAND's approach to protocol identification is fully automated; however, the system itself does not provide any useful method for automatically generating identifiers. Although this issue is out of the scope of this paper, we recognize that it is a serious deficiency that needs to be addressed.

The last issue is speed. Pattern matching in pysand is accomplished using Python's built-in substring finding function, which is, in Python 2.5, implemented as a variation of the Boyer-Moore algorithm and exhibits sublinear order ($O(n/m)$) in good cases and no worse than $O(nm)$. In practice, it works quite quickly. An even faster alternative to this might incorporate a Bloom filter, a probabilistic data structure capable of quickly determining set membership with the risk of false positives but incapable of producing false negatives [11], if the risk of false positives is judged acceptable.

Another speed concern is the overhead inherent in using libnids, which contains a full implementation of the Linux TCP/IP stack. Using libnids allows graceful handling of boundary cases that may confound other systems but perhaps introduces an overly large amount of overhead. Nevertheless, the graceful treatment of these boundary cases—and, in fact, all possible cases involved in TCP reconstruction—is one of the properties setting SAND apart from other systems.

Nevertheless, the speed of the system seems acceptable for many purposes. In a number of preliminary tests, rates between 30Mbps and 220Mbps were achieved on a moderately loaded AMD Athlon 64 X2 5000+ Linux system with 2GB of RAM. Various protocols and identifiers were included in these tests, the selection of which is expected to influence the performance of the system. Performance data and analysis is incomplete and likely to be the subject of future work.

A final set of examples of the advantages of the SAND approach over conventional TCP port-based identification schemes is illustrated by elaborating on an earlier comparison to Wireshark's identification abilities. An SSH server was configured to bind to port 5322, an instance of an alternate-port configuration suggested by some commercial web hosts, a web server was bound to port 22, and Wireshark was used to sniff the traffic to those servers.

In default configurations, Wireshark can immediately identify traffic on port 22 as SSH and on port 80 as HTTP; however, in this case Wireshark identified the SSH traffic on port 5322 only as "TCP", whereas it identified the web traffic on port 22 as SSH (Fig. 1). SAND, on the other hand, identified both connections after viewing 5 and 6 packets, respectively: that is, the three-way TCP handshake followed by the first few packets of each protocol's connection.

This striking, though manufactured, example stands by itself as an excellent argument for the usefulness of—and, in fact, need for—an alternate means of identifying traffic.

5. Conclusions and Future Work

This paper presents a new signature-based approach to the identification of protocols operating on TCP streams, dubbed SAND. Furthermore, it describes our reference implementation and its advantages and limitations. SAND is a signature-based solution for identifying protocols operating on TCP, performing full packet reassembly using libnids' reimplementation of the Linux TCP/IP stack. SAND operates on complete streams of traffic identifying protocols using sets of signatures.

This paper has hinted at directions for future work in this field. Specifically, a survey of real-world traffic, including attacks against similar systems, would prove valuable in confirming or refuting the need for the highly robust stream reassembly we implement.

The speed and performance of the system also needs to be analyzed in detail. The exact performance consequences of properties of the identifier (such as number and length of signatures) should be weighed alongside the properties of the traffic. Detailed results in that area will allow realistic judgments to be made about the feasability of deploying a fully stream-based network device in a real network.

If proven necessary, the performance of a conventional string-searching algorithm could be augmented by including an alternate string testing system; this approach is worth examining. For example, Dharmapurikar, *et al.* successfully utilized Bloom filters to improve performance in signature-matching field-programmable gate arrays [3].

We also intend to integrate SAND with network control systems; initially, we expect a sensible configuration to examine would be to augment port-based firewalling with content-based protocol *confirmation*, to more finely tune a firewall's rules or simply for quality of service decisions.

A SAND-like approach for UDP protocol detection needs to be developed; this has been less well-studied than TCP identification. Clearly a stream-based approach is inviable for a connectionless protocol like UDP; some useful substitute needs to be examined.

Finally, a word about the future of network protocol identification in general is warranted. This paper proposes and advocates for a new approach to identifying network protocols. It is our hope that reviving discussion of this necessary study will stimulate further work in the field, because the problem is only worsening, as protocols such as Skype and BitTorrent, which violate the basic assumptions that enable port-based identification to work, proliferate even further.

References

[1] S.A. Baset and H. Schulzrinne. An Analysis of the Skype Peer-to-Peer Internel Telephony Protocol. *Arxiv preprint cs.NI/0412017*, 2004.

[2] G. Combs et al. Wireshark, 2008. `http://www.wireshark.org/about.html`.

[3] S. Dharmapurikar, P. Krishnamurthy, T.S. Sproull, and J.W. Lockwood. Deep Packet Inspection using Parallel Bloom Filters. 2004.

[4] J. Levandoski, E. Sommer, and M. Strait. Application Layer Packet Classifier for Linux. 2008. `http://l7-filter.sourceforge.net/`.

[5] A. Moore and K. Papagiannaki. Toward the Accurate Identification of Network Applications. *PAM*, March 2005.

[6] B.C. Park, Y.J. Won, M.S. Kim, and J.W. Hong. Towards Automated Application Signature Generation for Traffic Identification.

[7] S. Sen, O. Spatscheck, and D. Wang. Accurate, scalable in-network identification of p2p traffic using application signatures. *Proceedings of the 13th international conference on World Wide Web*, pages 512–521, 2004.

[8] R. Wojtczuk. Libnids, 2008. `http://libnids.sourceforge.net/`.

[9] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Transport Layer Protocol. Technical report, RFC 4253, January 2006.

[10] S. Ertel. Unstructured P2P networks by example: Gnutella 0.4, Gnutella 0.6.

[11] A. Broder and M. Mitzenmacher. Network Applications of Bloom Filters: A Survey. *Internet Mathematics*, 1(4):485–509, 2004.

[12] Y.J. Won, B.C. Park, H.T. Ju, M.S. Kim, and J.W. Hong. A Hybrid Approach for Accurate Application Traffic Identification. *End-to-End Monitoring Techniques and Services, 2006 4th IEEE/IFIP Workshop on*, pages 1–8, 2006.