Network Security Through Conservation of Complexity

Scott C. Evans and Bruce Barnett

***Abstract*-- The problem of Network Security is approached from the point of view of Kolmogorov Complexity (see [2] for background). The principle of conservation of complexity is utilized to objectively identify healthy complexity norms and detect attacks via deviation of these norms under TCP/IP. Observed complexity changes that fall within expected bounds are indicators of system health, while complexity changes outside the expected bounds for normal protocol and application use are indicators of system fault or attack. Experimental results using FTP normal and attack sessions are presented.**

***Index Terms*— Network Security, Kolmogorov Complexity**

I. Introduction

UNITED States military operations have a long and rich history of exploiting technological advantage and engraining a deep technical understanding in every operator. US Navy nuclear propulsion plant operators for example, understand intimately the physical laws that govern the operation of their equipment. This proficiency in thermodynamics and propulsion in our military stands in sharp contrast to the science of information assurance in commercial or military operations. For example, it would be entirely unacceptable if the first indication of a heat exchanger leak in a nuclear powered ship were someone noticing that a rat had crawled inside of an empty tank. The operators would have noticed first a pressure drop, or reduced heat transfer performance and temperature changes. The leak would have been identified and corrective action taken long before any real damage had occurred to the mission or equipment. But many information security problems go unnoticed until extreme damage is done and an absurd result finally reveals the problem after the fact. We lack in the information assurance domain the physics of information that would enable operators to apply the same degree of diligence to their mission critical information networks that they apply to other mission critical systems.

Towards the goal of establishing a fundamental science of information assurance, Kolmogorov Complexity was proposed in [2] to be a fundamental property of information that can be used as a basic information parameter from which to build laws and models of information security. In this paper we review and expand upon the concepts introduced in [2] by developing the concept of Conservation of Complexity and applying it to the TCP/IP network protocol suite.

The advantage of using Complexity as a fundamental parameter to monitor system health and achieve information security lies in its objectivity. Any given string has a Kolmogorov Complexity without regard to the details of the system on which it is running. The operating system, protocol being used, and meaning of the data represented by a particular string, while related to string complexity, need not be known in order to measure string complexity. Kolmogorov Complexity is an inherent property of a string that can be used to build the science of information assurance in a way that is similar to role played by parameters such as pressure and heat in thermodynamics.

This paper applies the principle of conservation of complexity across network protocols as an objective means to benchmark normal behavior and detect network attacks. In order for this paper to be self-contained we briefly review Kolmogorov Complexity and the principle of conservations of complexity. Next we discuss the challenges that must be overcome in order to apply this principle on a network running TCP/IP. Finally we give experimental evidence showing the usefulness of this principle and its ability to detect FTP attacks by monitoring only the complexity of TCP/IP protocol data.

II. Background

A. Kolmogorov Complexity

Kolmogorov Complexity (K(x)) is a measure of descriptive complexity contained in an object or string (x). It refers to the minimum length of a program such that a universal computer can generate a specific string. A good introduction to Kolmogorov Complexity is contained in [1] with a solid treatment in [4]. Kolmogorov Complexity is related to Shannon entropy, in that the expected value of K(x) for a random sequence is approximately the entropy of the source distribution for the process generating the sequence. However, Kolmogorov Complexity differs from entropy in that it relates to the specific string being considered rather than the source distribution. Kolmogorov Complexity can be described as follows, where φ represents a universal computer (Turing machine), p represents a program, and x represents a string:

Manuscirpts submitted March 8th 2002. The authors are with GE Global Research Center in Niskayuna, NY. Lockheed Martin Systems Integration Owego, NY, funded this work, technically transitioning ideas developed under DARPA Information Assurance and Fault Tolerant Networks Projects contract F30602-01-C-0182 and managed by the Air Force Research Laboratory (AFRL) Information Directorate.

$$K_{\varphi}(x) = \left\{ \min_{\varphi(p)=x} l(p) \right\}.$$

Random strings have rather high Kolmogorov Complexity – on the order of their length, as patterns cannot be discerned to reduce the size of a program generating such a string. On the other hand, strings with a large amount of structure have fairly low complexity. Universal computers can be equated through programs of constant length, thus a mapping can be made between universal computers of different types, and the Kolmogorov Complexity of a given string on two computers differs by known or determinable constants.

The conditional Kolmogorov Complexity, K(y|x) of a string y given string x as input is described by the equation below:

$$K_{\varphi}(y \mid x) = \left\{ \begin{array}{c} \min_{\varphi(p,x)=y} l(p) \\ \infty, \textit{ if there is no } p \textit{ such that } \varphi(p,x) = y \end{array} \right\},$$

where l(p) represents program length p and φ is a particular universal computer under consideration.

The major difficulty with Kolmogorov Complexity is that it is not computable. The length of any program that produces a given string is an upper bound on the Kolmogorov Complexity for this string, but you can't compute the lower bound [4].

B. Estimators of Complexity

As discussed above, exact measurement of Kolmogorov Complexity is not achievable, however various methods of estimating complexity are available. A natural choice for estimation of complexity is the class of universal compression techniques. In [27] Lempel and Ziv define a measure of complexity for finite sequences rooted in the ability to produce these sequences from simple copy operations. The familiar Lempel Ziv 77 and 78 [29], [30] universal compression algorithms (LZ77, LZ78) harness these principles to yield compression algorithms that can approach the entropy of an infinite sequence produced by an ergodic source. Unix compress, which is based on the LZ78, is used as the compression estimation algorithm in this paper.

III. Conservation of Complexity

Conserved variables enable one to deduce parameters from the presence or absence of other parameters. The Law of Conservation of Matter and Energy for example allows one to deduce how well a thermodynamic system is functioning without knowing every parameter in the system. Heat gain in one part of the system was either produced by some process or traveled from (and was lost from) another part of the system. One knows that if the thermal efficiency of a thermodynamic system falls below certain thresholds then there is problem. On the other hand, if more heat is produced by a system than expected, some unintended process is at work. A similar situation is desirable for information systems – the ability to detect lack of assurance by the presence of something unexpected, or the absence of something that is expected. This seems to be far from reach, given that information is easily created and destroyed with little residual evidence or impact.

Since complexity of a given string can only change through computational operations, it is in a sense a conserved variable. Suppose the exact Kolmogorov Complexity K(*S*) of a string of data *S* was available. One would essentially have a conserved parameter that could be used to detect, resolve or infer events that occur in the system, just as tracking heat in a thermodynamic system enables monitoring of that system. Operations that affect string *S* and cause it to gain or lose complexity could be accounted for, and an expected change in complexity should be resolvable with the known (secured) operations occurring in the information system to produce expected changes in complexity. Complexity changes that occur in a system that cannot be accounted for by normal system operations indicate unauthorized processes taking place. Thus, in the ideal case where Kolmogorov Complexity is known, a check and balance on an information system that enables assurance of proper operation and detection of unauthorized activity is possible. Unfortunately (as previously discussed) a precise measure of Kolmogorov Complexity is not computable. We can, however, bound the increase in Kolmogorov Complexity as shown in the theorems below.

A. Theorems of Conservation

Kolmogorov Complexity, K(x), can be thought of as a conserved parameter that changes through computational operations conducted upon strings. In order for K(x) to be a conserved parameter one must account for changes in K(x). Two theorems are presented below that enable bounds to be placed on the changes in K(x) that occur due to computational operations occurring in an information system. The two theorems, which are proven in the appendix, show bounds on the amount of complexity that can exist due to knowledge of other strings or computational operations. These theorems are proven in the appendix.

1) Theorem 1: Bound on Conditional Complexity

$$K_{\varphi}(y \mid x) \leq K_{\varphi}(y)$$

2) Theorem 2: Bound on Complexity Increase Due to a Computational Operation

$$K_{\varphi}(y \mid x, p) \leq K_{\varphi}(x) + L(p)$$

B. The Principle of Conservation of Complexity

As shown above, while not computable from below, upper bounds on the increase in Kolmogorov Complexity can be crudely known by keeping track of the size of programs that affect data. This bound may be incredibly loose, as it is quite

possible to operate on a string and make it much less complex than the input. One would need a method to recognize this simplification. However, these results provide an intuitively attractive method for quantifying the "work" performed by a computational operation on information – the change in complexity introduced by the operation. A thorough treatment of bounds related to K(y|x) and the "Information Distance" between strings is contained in Bennett et al. [28].

The principle of conservation of complexity can be applied to closed as well as open information systems in the same manner that thermodynamic systems are applied to closed or open systems. It is not necessary to maintain an account of every operation that takes place in an information system to utilize this principle. Expected complexity changes through authorized processes that are either mathematically determined or measured and benchmarked for healthy systems can be applied to various points in the information system in the same manner that temperature meters and mass flow indicators monitor the health of open thermodynamic systems.

The principle of conservation of complexity is summarized in figure 1, where the complexity inherent in a stream of data over time falls within bounds determined by the authorized processes of the system or protocol. This principle can be applied to any process for which finite sets of authorized processes are known or measurable. An ideal application is in network protocols. See [6] for a case where complexity is used to detect Distributed Denial Of Service (DDoS) attacks based on complexity of packet data. In this paper, without considering the data payload we look at the transport layer protocol alone. As information transitions across each layer of network protocol stack and messages are exchanged within the policies of the protocol, finite changes in complexity occur. Expected behaviors can be either derived from the protocol rules and policies or measured and benchmarked for healthy systems.

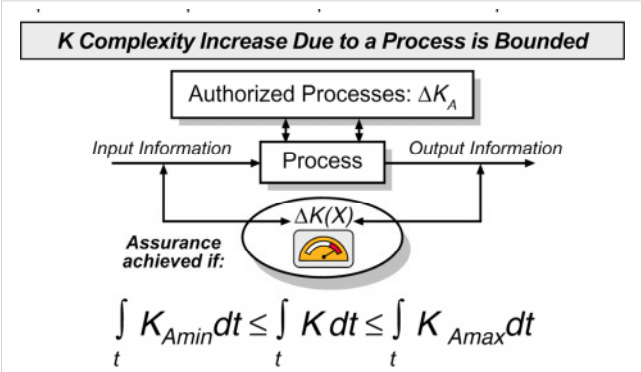

Figure 1: Conservation of Complexity

IV. Conservation of Complexity in Network Protocols

A. Challenges

The major difficulties in applying conservation of complexity to a protocol lie in both coordinating the timing aspects of the protocol and in dealing with the wide variation in operations inherent in normal protocol use. Packets traversing a protocol stack may be routed through different host computers - acknowledgements may be lost, retries may be attempted. However, from various points in the network a stream or flow of data between two host ports, for example, may be measurable. A firewall is a logical place for this to take place, since routing ambiguity is lost. At a measurement point, timing aspects of the protocol may be kept track of by simply concatenating session packets together into a single string. Two separate strings may be maintained – one for each direction, or a single combined string may be maintained. When concentrating on the protocol alone, the application specific data portion of the packets may be ignored (unless application behavior is well known) in order to restrict the operations that can occur on the data to a known and manageable set.

B. Conservation of Complexity Across TCP/IP

While networks are not closed systems, network protocols provide bounds on the operations and expected behavior that can or should occur when properly using a protocol. Wide variation in complexity will normally occur at the application layer. However, since the very nature and purpose of protocols is to enforce regularity, it is not unexpected that restricted complexity changes occur in the normal use of protocols. Thus the following hypothesis is introduced: The principle of conservation of complexity applied to network protocols – using either calculated or measured norms – can identify healthy or attack behavior. The results in the next section show that complexity metrics applied to transport layer header information of FTP sessions can distinguish attack vs. normal protocol use. We utilize UNIX compress to measure complexity, as noted in section II. The results show that complexity of protocol use in various attack scenarios is typically and discernibly less complex than normal FTP sessions.

V. Experimental Results

The Conservation of Complexity principle is tested using FTP[8] sessions between a client and server node.

A. Experimental Setup

All tests were performed using the same two systems. The client was running Redhat Linux 7.1. The server was running Solaris 2.6. The *tcpdump* command [9], running on the server, was used to capture all traffic.

B. Normal Traffic Patterns Generated

To distinguish between typical FTP traffic and attack traffic,

we first specified several typical FTP sessions, using a variety of functions and tools. The following typical sessions were specified:

1. Incorrect login
2. Connected and left
3. Connected, looked for a file, and left
4. Put one file
5. Retrieved one file
6. Retrieved several files
7. Retrieved an entire directory
8. Used a web browser to examine and obtain several files.

The first six sessions were executed using *expect[10]*. The seventh was using *ncftp[11]* – often used to get multiple files; the last session was executed using Netscape Navigator.

C. Attack Patterns generated

We found several FTP exploits on the Internet. These exploits are summarized in table 1. We selected all that could be compiled and made to run in a script. These exploits are written in various styles and techniques. This provided a variety of implementations, as the source language, target, and exploit technique varied. Techniques included buffer overflows, Trojan executables, and denial of service attacks. At least three different implementations of FTP daemons are targeted. Some succeeded on a Solaris server, such as techniques to break out of a *chroot*() jail. Four of the exploits were denial of service attacks written in Perl. Some exploits targeted the same weakness, but the code was written in a different style. The *woot-exploit*[24] attack launched 978 FTP sessions before we terminated it. The *wuXploit[26]* attack launched two different FTP sessions.

Name	Program	Version	Type of attack
Babcia[13]	proftpd	1.2.0	Realpath(): overflow
Duke[14]	Wu-ftpd	2.4.b18	Realpath(): overflow
wh0a[15]	Wu-ftpd	2.4.b18	Realpath(): overflow
w00f[16]	Wu-ftpd	2.4.18	Realpath(): overflow
Bobek[17]	Wu-ftpd	2.6.0	SITE EXEC metacharacters
Ftpwarez[18]	Wu-ftpd	2.4 b17	Realpath(): overflow
Wftpd241[19]	WFTPPro	2.4.1	RNTO w/o RNFR DoS
Wftpd241-11-1[20]	WFTPPro	2.4.1	STAT send before LIST completes, DoS
Wftpd241-11-2[20]	WFTPPro	2.4.1	REST STAT before STOU, DoS
Wftpd241-11-4[20]	WFTPPro	2.4.1	MLST before LOGIN, DoS
Wu-ftpd26[21]	Wu-ftpd	2.6.0	SITE EXEC metacharacters
Wuftpd-god[23]	Wu-ftpd	2.6.0	MKD/CWD:overflow
Wu-ftpd-v2.4.4[24]	Wu-ftpd	2.4.4	SITE EXEC
Woot-exploit[25]	Wu-ftpd	2.6.1	SITE EXEC file glob
WuXploit[26]	Wu-ftpd	2.6.1	Auto-compress spoof

Table 1

D. Measurement method

Tcpdump was used to convert the packet trace into ASCII. This string of information corresponds to a network session. The string representing the session was compressed using Solaris's standard *compress* utility. Each FTP connection resulted in a single point on the graph.

The first examination of the data showed no pattern. It was felt that there was significant noise in the network trace. We then presented the data in a form that would hopefully allow patterns to be more apparent.

Normal network traces consists of traffic flowing both ways. We felt that if we separated the traffic into two sets – (outgoing and incoming) and then appended them, the patterns would be more apparent. Therefore our traces consist of all of the packets from the client, followed by all of the packets from the server. However, each trace has a lot of data that has large effects on data complexity that are not directly related to the regularities associated with the protocol. For example, every line has a unique timestamp twice. To reduce this "complexity noise", we used a filter to reduce the trace data to the following TCP information:

- Hostnames and ports (identical for all cases)
- Flags
- Bytes transmitted
- Bytes acknowledged
- Available window size

Examples of raw vs. filtered *tcpdump* data are contained in the appendix.

Comment 1: We wish to emphasize that the data examined only consists of TCP-specific information. No analysis of the data within the payload was used. Also, we only analyzed port 21 traffic, which is used as a control channel for the data. We did not examine the complexity of the data channel.

Comment 2: The filtering and rounding of the TCP data is justified due to the fact that our interest is in monitoring and benchmarking the normal complexity of the protocol. Integers and numbers involved in timestamps etc. are not related to protocol complexity and thus can and should be removed as they add noise to the data that is unrelated to the healthy use of the protocol.

E. Experimental Measurement Results

There were 8 typical FTP sessions, and 996 attacks. Figure 2 shows a detail of the original data set, without filters. The X-axis is the size of the trace in bytes, and the Y-axis shows the size of the compressed trace.

The ratio of these two values shows the complexity of the trace. Some of the samples coincided and are not visible. Attacks and normal FTP sessions followed the same complexity curve. However, there was no clear distinction between the two sets.

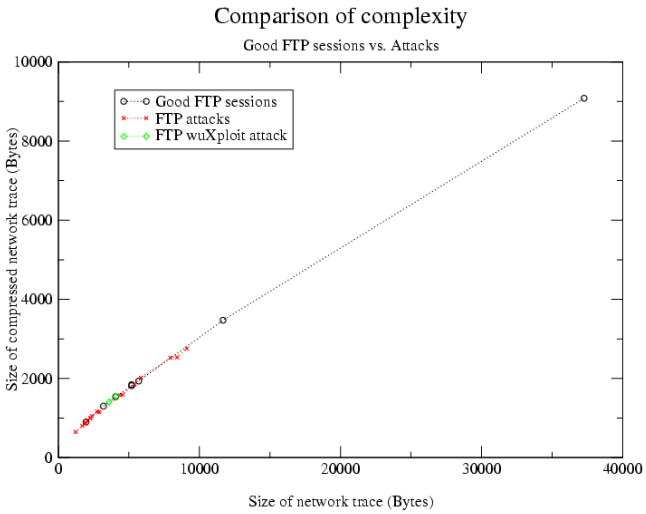

Figure 2: Size of network trace vs. compressed network trace of healthy and attach session, unfiltered data.

Figure 3 shows the entire data set after applying the filtering. Figure 4 shows the same dataset in detail

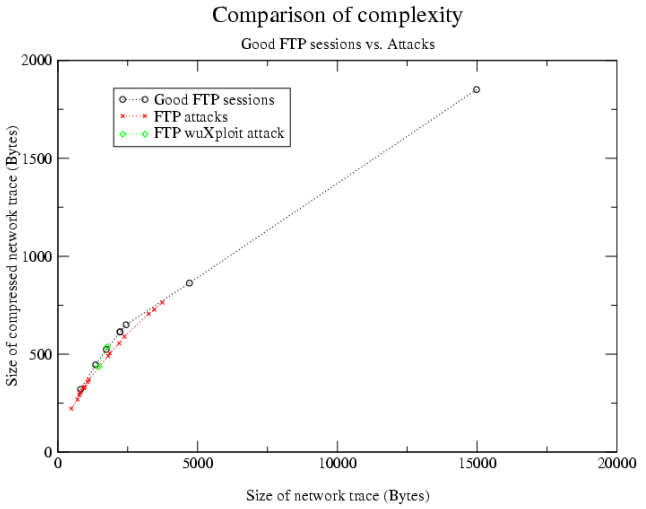

Figure 3: Size of network trace vs. compressed network trace of healthy and attach session, filtered data.

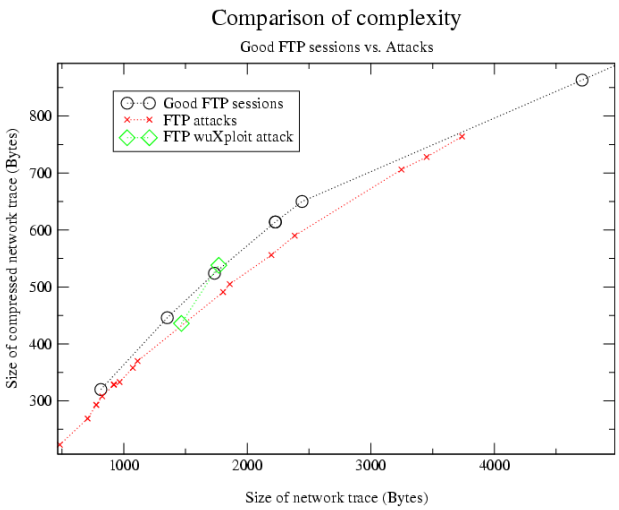

Figure 4: Size of network trace vs. compressed network trace of healthy and attach session, filtered data – detailed view.

F. Analysis of results

The results indicate that FTP sessions have predictable complexity. The complexity curves generated by this data are not linear initially, but approach a linear compression ratio of approximately 4:1 as the trace size increases. This could be do in part to the nature of LZ78 to reach asymptotic limits as file size increases – the uniqueness individual protocol sessions is expected to be less measurable by LZ78 as session length increases. Before we applied our filters, typical sessions had the same complexity as attacks. After filtering, the distinction between these two classes of connections became remarkable: attack sessions have less complexity as indicated by a higher compression ratio of 4.25:1 vs. healthy sessions with a compression ratio of 3.75:1. The curves are smooth and consistently show that attack sessions are more compressible, hence less complex than normal sessions.

Statistical analysis of the data comprising filtered traces less than 4000 Bytes confirms a significant difference between normal and attack complexity curves. We fit a linear regression of log(compressed filtered trace size) vs. log(filtered trace size) on the attack series due to the exponential nature of the curves. The F-statistic indicates a highly significant fit for which the p-value is less than 0.0001. The closest piece of normal traffic differed from this attack model by 4 sigma (99.99%).

One attack (wuXploit) is highlighted in Figure 4 as two points indicated by squares. In this attack, two sessions are initiated. The first session sets up the system to be exploited by creating a Trojan compression program. The second session is a normal session, simply requesting a compressed file. As indicated on Figure 4, the active attack is on the exploit curve, and the normal file transfer is on the normal curve, thus validating the complexity metric.

Several reasons may be used to explain why FTP exploits were measured to be less complex at the TCP level than normal traffic. One possible reason is that exploits tend to send more data to the server without analyzing the results. Another is that standard applications are designed to be flexible, and attempt to negotiate protocol extensions if possible. This makes the session more complex. Thirdly, people who write exploits tend to simplify the complexity, often combining steps. For instance, rather than sending the USER and PASS parameters in two packets, they tend to use one request, and one packet.

VI. CONCLUSIONS

The results show that the principle of conservation of complexity applied to network protocols holds promise with respect to objectively benchmarking network health and identifying network attacks. There is a strong distinction between typical and attack complexity characteristics in the FTP data set we used when analyzing TCP-specific information that has been filtered to remove complexity noise unrelated to the protocol itself. These results show that TCP-based protocols have predictable complexity curves. That is,

complexity is conserved and falls within expected norms for normal protocol use. Different applications may have different curves, and the exploits we used apparently have remarkably different complexity curves from typical traffic. Therefore it is possible to use complexity to assist in distinguishing between typical traffic and exploits.

Future work is necessary in numerous areas to validate this principle. Similar experiments are necessary with larger data sets, different protocols and network configurations, and alternative measures complexity in order further validate and determine the bounds of usefulness for the principle of conservation of complexity. In particular, examining application specific commands, arguments, and data, both as individual sessions, and across multiple sessions, should be investigated.

APPENDIX

1) Theorem 1: Bound on Conditional Complexity

$$K_\varphi(y \mid x) \le K_\varphi(y)$$

Proof: Since K(y) is the minimal length program that produces string y with no input, input x can only reduce the length of the program required to produce y. At worst, x can be ignored completely, in which case K(y|x)=K(y). However, knowing x may reduce the program to produce y, depending on the extent that string x contributes towards generating string y or enables a more efficient generation of y. QED

2) Theorem 2: Bound on Complexity Increase Due to Computational Operation

$$K_\varphi(y \mid x, p) \le K_\varphi(x) + L(p)$$

Proof: Program p of length L(p) takes input string x to produce output string y. Proof by contradiction: consider a program p that could be run on input string x to produce string y. Assume that the complexity of y = K(y|x,p) > K(x) + L(p). But one could produce string y by first forming string x with program of length K(x), then running program p of length L(p), thus producing y with a program of length K(x) + L(p). But this violates the definition of Kolmogorov Complexity as being the minimum length program, since a program of smaller length has been found. Thus the assumption is false and K(y|x,p) must be <= K(x) + L(p). QED

Example of Raw TCP Dump Data

```
12:25:01.676332 3.1.175.241.32827 > 3.1.4.53.21: S 1887323655 : 1887323655(0) win 5840 <mss 1460, sackOK,timestamp 776798 0, nop, wscale 0> (DF)
12:25:01.677773 3.1.175.241.32827 > 3.1.4.53.21: . ack 4059525913 win 5840 <nop,nop,timestamp 776799 2271017432> (DF)
12:25:01.748597 3.1.175.241.32827 > 3.1.4.53.21: . ack 60 win 5840 <nop,nop,timestamp 776806 2271017503> (DF)
```

Example of Filtered TCP Dump Data:

```
3.1.175.241 * 3.1.4.53 ftp: S 0 win 5840
3.1.175.241 * 3.1.4.53 ftp: . ack 0 win 5840
3.1.175.241 * 3.1.4.53 ftp: . ack 60 win 5840
```

ACKNOWLEDGMENT

We wish to Steve Bush, Amit Kulkarni, and Colin McCulloch for help and encouragement in developing these ideas.

REFERENCES

[1] Cover, T. M. and Thomas, J. A. Elements of Information Theory. Wiley, NY, 1991
[2] Evans, S, Bush, S. F., and Hershey, J., "Information Assurance through Kolmogorov Complexity", DARPA Information Survivability Conference & Exposition II, 2001, Proceedings Vol 2, pp 322-331.
[3] Kieffer, J. C. and Yang, E. "Sequential Codes, Lossless Compression of Individual Sequences, and Kolmogorov Complexity," IEEE Transactions of Information Theory, Vol 42, 1 January 1996
[4] Li, M. and Vitányi, P. An Introduction to Kolmogorov Complexity and Its Applications, Springer, NY 1997
[5] Evans, S. C., Hershey, J. E "Sequence Complexity Probes," submitted to SCI conference in Orlando, July 2002.
[6] Kulkarni, A. B., Bush, S. F. and Evans, S. C. "Detecting Distributed Denial-of-Service Attacks using Kolmogorov Complexity Metrics," GE Research Technical Report 2001CRD176. December, 2001.
[7] Evans, S. C. and Bush, S. F. "Symbol Compression Ratio for String Compression and Estimation of Kolmogorov Complexity," GE Research Technical Report 2001CRD159, November 2001.
[8] FTP - http://www.ietf.org/rfc/rfc959.txt
[9] Tcpdump – see http://www.tcpdump.org/
[10] Expect - http://expect.nist.gov/
[11] Ncftp - http://www.ncftp.com/
[12] Netscape Navigator http://www.netscape.com/download
[13] Babcia - http://security-archive.merton.ox.ac.uk/bugtraq-199908/0396.html
[14] Duke http://lists.nas.nasa.gov/archives/ext/linux-security-audit/1999/03/msg00174.html
[15] Wh0a http://online.securityfocus.com/archive/1/12962
[16] Woof - http://security-archive.merton.ox.ac.uk/bugtraq-199905/0022.html
[17] Bobek - http://v.freebsd.lublin.pl/sources/security/security/bobek.c
[18] Ftpwarez - - http://www.spitzner.org/winwoes/packetstorm/0003-exploits/ftpwarez.c
[19] Wftpd241 - http://www.mp3glowe.com/defson/files/hacking/exploits/wftpd241.txt
[20] Wftpd-241 http://www.securiteam.com/exploits/5OP0P0K20E.html
[21] Wu-ftpd26 http://security-archive.merton.ox.ac.uk/bugtraq-200007/att-0046/01-wu-ftpd26.c
[22] Wu-ftpd-exp http://online.securityfocus.com/archive/1/68687
[23] Wuftpd-god http://online.securityfocus.com/attachment/2002-02-26/wuftpd-god.c
[24] Wu-ftpd-246 http://invaultech.com/files/Hacks2/wu-ftpd-v2.4.4.c
[25] Woot-exploit http://online.securityfocus.com/archive/82/244938
[26] WuXploit http://open-security.org/vulnerable/servers/ftpd/wuXploit.tgz
[27] Lempel, A. and Ziv, J. "One The Complexity of Finite Sequences," IEEE Transactions of Information Theory, Vol IT 22, January 1976, pp 75-81.
[28] Benett et. Al "Information Distance" IEEE Transactions on Information Theory, Vol 4, no 4. July 1998, pp 1407-1423.
[29] Ziv, J. and Lempel, A. "A universal algorithm for sequential data compression," IEEE Trans. Inform Theory, vol IT-23, pp. 337-343, 1977.
[30] Ziv, J. and Lempel, A. "Compression of individual sequences via variable length coding," IEEE Trans. Inform. Theory, vol IT-24, pp. 530-536, 1978.