Global Information Assurance Certification Paper

GCIA practical version 3.3 for Andrew R. Jones

Part I - Using Randomness Tests as an Intrusion Detection Technique

Abstract

This paper will examine the feasibility and applicability of using a suite of randomness tests (that is, statistical tests to determine how close a series of numbers is to being random) to analyze large quantities of network traffic and find anomalous data packets.

Introduction, background, and presentation of tests

One of the issues that intrusion detection systems have always had to deal with is the fact that most of them are signature-based, and therefore cannot detect new attacks until a signature is written for them. For those intrusion detection systems that do include algorithms to detect unknown attacks, the question has always been what to look for -- that is, how to characterize packets as "anomalous" or "normal". To boot, it has to be fast.

Inspired both by Bruce Schneier's Applied Cryptography and by stories of the Demon Net, an ISP in the UK that for years sent mangled packets to other computers, apparently because of a hardware problem, the idea of using randomness tests to help determine what packets meet the definition of "anomalous" occurred to this author. The hypothesis is that normal Internet traffic will have a certain level of randomness according to a series of tests, and that outliers will be packets that need to be examined.

To come closer to the exact experiment specification, we need to examine some requirements and limitations. "Internet traffic" includes an extremely wide variety of kinds of data, and it would be unreasonable to think that all normal Internet traffic could be folded under one umbrella randomness rating. On one end of the spectrum, we have SSH and IPsec, both of which are well encrypted (meaning highly random), and on the other end we have data in HTML format or in plaintext (highly structured). For that reason, it would be best to attempt classification based on port, or failing that, protocol. Even there we expect to have problems with certain ports. Port 80 will be difficult no matter what we try. All kinds of data are transported over port 80, and to make things more complicated, some Web servers and browsers are capable of compressing Web pages before transferring them.

Another limitation that we impose from the very beginning is that we will work only at the application layer, and we will not try to apply this technique to the IP or upper layer protocol headers. It might be interesting to try, but it is probably not a good application of the technique. For one thing, the headers are small enough that this is better done with a signature-based system. For another thing, there may well not be enough data to obtain meaningful results. Some of the tests that we will be examining (but not the ones that we will end up using) require a large minimum number of input bits (in some cases, on the order of 10^6 bits).

Some background to randomness is called for. What is randomness, and where do we get it? A random series of bits has a number of properties, the most important being that it cannot be predicted. For the purpose of cryptography, Bruce Schneier has this to say: "The primary point is to generate a sequence of bits that your adversary is unlikely to guess." (Schneier, p. 422) As we will see in a moment, random sequences satisfy certain properties, such as:

- A proportion of ones to zeros that is very close to 1:1
- The number and length of runs of ones or zeros is what one would expect, according to probability theory
- The sequence is not compressable
- The sequence exhibits no structure, such as linear dependence

From this list, we can already see where certain kinds of data will fall. Compressed data will have a high degree of randomness, going on the assumption that the compression algorithm is good, and thus that the output of same is not further compressable. Encrypted data will also appear highly random, because the purpose of encryption is to make data appear random so cracking the encryption is more difficult. On the other hand, English text has about 1.3 bits of information per byte, giving us very low randomness. (Schneier, p. 15)

The US National Institute of Standards and Technology produced a series of randomness tests for the purpose of testing pseudorandom number generators to see if they are suitable for use in cryptographic applications. This project was carried out by the Computer Security Division and the Statistical Engineering Division of NIST, and the resulting C code and documentation has been placed in the public domain with the request of recognition given where used. The stated goals of the project are:

1. to develop a battery of statistical tests to detect non-randomness in binary sequences constructed using random number generators and pseudo-random number generators utilized in cryptographic applications
2. to produce documentation and a software implementation of these tests
3. to provide guidance in the use and application of these tests (Computer Security Division, NIST)

A description of each test in the test battery follows, with some commentary as to time complexity, applicability to the current task, and anything else deemed relevant. Time complexity is one of the most important factors in deciding which tests have a future in intrusion detection and which don't. An IDS has to be fast, and an algorithm that does not have a linear time complexity (or better) is not acceptable. In all cases, n denotes the number of input bits to the algorithm. For a detailed description of the tests, please see the original documentation from NIST. (Rukhin et al)

1. **Frequency (Monobit) Test**: This does nothing more than count the number of zeros and ones in the sequence. The time complexity is O(n). This test could be useful, but the next test is better.

2. **Frequency Test within a Block**: This is the same as the previous test except that the input stream is partitioned into equally sized blocks, and the monobit test is run for each block. This is done by changing all zeros to -1's and adding all of the bits in the block to be tested. This algorithm also has a time complexity of O(n). This test was included because the input stream could be random according to the previous test, but locally very predictable. A sequence of 1000 zeros followed by 1000 ones would pass the previous test but fail this one. The important aspect of this test for this paper is that it will give us a pretty graph. Seriously, we expect to have fluctuations in the randomness of real Internet traffic (HTTP transfer of an encrypted file, for instance -- the HTTP headers are all plaintext and therefore highly predictable, while the file is highly random), and we need to be able to see those to build a profile of what certain kinds of traffic should look like.
3. **Runs Test**: This test ascertains the number of runs in the sequence, or, put another way, how often the sequence fluctuates between zero and one. The time complexity is O(n).
4. **Test for the Longest Run of Ones in a Block**: This test records the number of occurrences of runs of ones of certain lengths, somewhat contrary to what one would expect from the name. This test also divides the input stream into blocks. The time complexity is O(n). This test can also provide us with data partitioned over a number of possibilites.
5. **Binary Matrix Rank Test**: This test builds matrices from the input bits and performs row reduction on the matrices to determine the rank of the matrices. A matrix with a high rank indicates linear independence, which in turn is indicative of randomness. The time complexity of this test with a constant matrix size is O(n). If the matrix size is made to vary with the number of input bits, it gets much more complex. If one dimension of the matrix is m (the matrices are always square in this test), the time complexity of the row reduction algorithm in machine instructions (not in elementary row operations) is $O(m^3)$. Regardless of whether the matrices are variable-sized or not this test is very obviously computationally expensive.
6. **Discrete Fourier Transform (Spectral) Test**: This test performs a Fourier transform on the input data to identify periodicities. The author is not familiar with Fast Fourier Transforms, and as a result cannnot evaluate the applicability of this algorithm. What can be said is that in precursor tests, this test used the entire 512MB of RAM and 1GB of swap in the author's computer before the operating system was forced to kill the process. That put it out of the running.
7. **Non-overlapping Template Test**: This test attempts to find instances of a certain pattern in the input data. The data are tested using successive bits as the starting position until the pattern is found. At that point, the starting point is moved along the sequence by the number of bits in the pattern. The time complexity of this test is O(n).
8. **Overlapping Template Matching Test**: This test is similar to the one above. The only difference is that the starting position for the test is always moved by one bit, regardless of whether the last starting position produced a match. The time complexity of this test is also O(n).

9. **Maurer's "Universal Statistical" Test**: This test attempts to determine the compressability of the input stream by measuring distances between reoccurring bit patterns. The time complexity is O(n). This test is variously called Maurer's test and the universal statistical test in this text.
10. **Lempel-Ziv Compression Test**: This test is also based on the compressability of the input sequence. It builds a "dictionary" of bit patterns that it finds. Assuming a binary tree implementation, the worst case time complexity is $O(n^2)$, while the best case is $O(n\log_2 n)$.
11. **Linear Complexity Test**: This test builds the smallest linear feedback shift register necessary to reproduce the input sequence. Linear feedback shift registers are a means of creating a pseudo-random number generators. The assumption is that a more complex linear feedback shift register implies randomness. This algorithm relies on other algorithms and mathematics with which the author is not familiar.
12. **Serial Test**: This test determines the frequency of all overlapping m-bit patterns in the sequence. This test is O(n), but it is obviously very computationally expensive. Testing all 2^m patterns over the entire input sequence will take time.
13. **Approximate Entropy Test**: This test is almost the same as the last one, except that it does the same thing again for all m+1-bit patterns and compares the results to those for the m-bit tests. This takes even more time than the last test.
14. **Cumulative Sums (Cusum) Test**: This test, like some of the other tests, adjusts the input sequence to make 0=-1, then adds up the partial sums of the sequence. This test looks for the maximum distance from zero that the partial sums assume. For a random sequence, this should be small. The time complexity of this test is O(n). This test as it stands does not work for our application. A possible adjustment to make it applicable to the current task would be to assume that there will be a bias of either ones or zeros, yielding a steadily, though erratically, increasing or decreasing sequence of partial sums. If linearity of that resulting sequence is assumed, a linear regression fit could be used as a characteristic of the input sequence. The slope of the line and the correlation would be the two useful pieces of data. Unfortunately, calculating a linear regression is a lot of work, making this test unacceptable for our requirement of high speed. A simpler idea that works for visual inspection is to simply plot the graph of the partial sums.
15. **Random Excursions Test**: This test measures the number of cycles in the random walk described for the last test in which each state is visited exactly K times. First of all, the last test created a random walk, defined by the sequence of partial sums. Second, a cycle is a partial sequence that begins and ends with zero. Third, K takes on the values zero to five in the test as delivered. Again, this would have to be altered to assume non-randomness, and probably linearity of the sequence of partial sums. The time complexity of this test is also O(n).
16. **Random Excursions Variant Test**: This test is much easier than the last one. It only tests for the number of times state x is visited in the random walk, where x takes on the values -9 to -1 and 1 to 9. The time complexity is still O(n), and the comments for the last two tests still apply.

After examining all of the possibilities, and considering the time constraints, four tests were chosen to begin with. Those tests are the block frequency test, the longest run of ones test, the runs test (that is, the number of runs), and Maurer's universal statistical test. These were chosen because they are easy to adapt from the code that the NIST group wrote, they were felt to be likely to be fast, they should provide meaningful results, and they are not too similar to each other (like the last three tests in the full battery from NIST are).

Some further explanation of the four tests is required. For the runs test, the total number of runs in the sequence is divided by the number of bits in the sequence, giving a normalized number of runs. The block frequency test automatically adjusts the size of the blocks to be tested to achieve between 500 and 5000 blocks, if possible. The size of the blocks does not go below 80 bits or above 800000 bits. The longest run of ones test was completely rewritten to test both for runs of zeros and of ones. It also keeps a tally of the numbers of runs of zeros and ones of all lengths found in the data set, in contrast to NIST's original test. NIST's test just looked for a few run lengths, then had two more categories for "less than the smallest length we test for" and "greater than the largest length we test for". At the end of the test, the tallies for each run length are divided by the number of runs of that digit (zero or one, that is) in order to normalize the results. Maurer's universal statistical test works by partitioning the input sequence into blocks, creating a table large enough to hold all 2^m possible m-bit patterns for a block, initializing the table, and calculating distances. The table is initialized by sequentially examining the first Q blocks and placing the block number in the table next to the bit pattern found in that block. The test statistic is calculated by examining the remaining K blocks, and for each block calculating the distance in blocks from this occurrence of the bit pattern found in the current block to the last occurrence, as kept in the table. The table is updated with the new block number and $\log_2$ of the calculated distance is added to a running sum. This sum is divided by K at the end of the test in order to normalize the result. The length of the blocks and the number of initialization blocks are calculated dynamically based on the number of input bits. A low number means high compressability or low randomness.

Analysis

In this section, we first present the reader with a large number of results from tests and skeletal interpretations, so that we can dissect the results in greater detail at the end and consider them all together. It needs to be mentioned that the graphs for the longest runs test are logarithmic with a base of 2. This was done to ensure that as many of the bars were visible as possible. It is also much easier to understand on a intuitive level with a logarithmic scale for the y axis.

In the first set of results, we see an FTP session for downloading a gzip-compressed tar file of C source code (the Cyrus SASL library, for what it's worth). We note that the FTP command channel is probably highly compressable because of the low result for Maurer's test, and that the majority of the blocks lie well below the axis for the block frequency test, indicating more zeros than ones. This then gives us our first indication as to how text is going to look according to our measurements. We need to be careful, though, because

this sampling of text data is very small. The gziped file, on the other hand, is not very compressable, which is to be expected. We also see that the majority of the bits by far are ones. We can see something of the structure of a gziped file in what looks like a header at the beginning and possibly a footer at the end. These parts have visually distinct properties in the block frequency graph. The graph of run lengths tells us that compressed data are likely to look like a linear decrease in the frequency of increasing run lengths, which corresponds to an exponential decrease in reality, since we are dealing with a logarithmic scale.

FTP command session #1
Size: 619 bytes
Normalized number of runs: 0.525444
Result of universal statistical test: 0.757331

FTP data session #1
Size: 5625195 bytes
Normalized number of runs: 0.495945
Result of universal statistical test: 10.159898

This test is another FTP session of a different gziped tar file of C code (MySQL, in this case). The FTP command channel looks very much like the last one. The FTP data channel, despite the fact that the file being transferred ought to have similar characteristics, looks very different in the block frequency graph. The range of the values in the block frequency test are much wider than in the previous graph, and they are not nearly as uniformly positive. The presence of the pattern between about blocks 900 and 1100 is disturbing, because it doesn't seem to fit with the rest of the data in the graph. The results for the other tests are about the same as the ones from the last FTP session.

FTP command session #2
Size: 898 bytes
Normalized number of runs: 0.526169
Result of universal statistical test: 0.757218

FTP data session #2
Size: 12133326 bytes
Normalized number of runs: 0.499128
Result of universal statistical test: 11.101890

This test is of an FTP transfer of a .exe (setiathome_win_3_07.exe). The high result for Maurer's test is a bit surprising, because it implies that .exe files are not very compressable. This may well be the case, though. The graph of block frequency is amazingly clear and reveals much of the inner structure of an executable. The graph of the run lengths is also very different from what little we have seen so far. There are more long runs, and there are longer runs, including very long runs of zeros far out onto the right side of the graph. We will see these properties again later.

FTP command session #3
Size: 643 bytes
Normalized number of runs: 0.522551
Result of universal statistical test: 0.753646

FTP data session #3
Size: 792064 bytes
Normalized number of runs: 0.491036
Result of universal statistical test: 8.090774

Here we see an MP3 being transferred over FTP. The MP3 is the theme song to OpenBSD 3.0. Not surprisingly, it is not very compressable, since the MP3 encoding scheme already does compression. The run length graph looks downright erratic, and it doesn't tell the whole story. Not pictured in the graph is a run of ones of length 2024 bits. There are many more long runs in these data than we have seen so far, and the reemergence of a peak at about 80 bits followed by a gradual decline is a surprise, along with many other smaller aspects of the picture.

FTP command session #4
Size: 603 bytes
Normalized number of runs: 0.515340
Result of universal statistical test: 0.749384

FTP data session #4
Size: 2898547 bytes
Normalized number of runs: 0.493045
Result of universal statistical test: 10.037311

Here we have the theme song from OpenBSD 3.1. It looks very similar to the previous result in most respects. It is somewhat more compressable, it tends more towards zeros than ones, and the range in the block test is not quite as wide as for the 3.0 theme song. The run lengths graph once again shows a strong tendency to lean more on the right side of the graph than other kinds of data, but this file lacks the run length features of the last that were so interesting.

FTP command session #5
Size: 603 bytes
Normalized number of runs: 0.518657
Result of universal statistical test: 0.752415

FTP data session #5
Size: 2912757 bytes
Normalized number of runs: 0.503963
Result of universal statistical test: 9.984847

This is the theme song for OpenBSD 3.2 (the author wasn't sure where else to find a good repository of legal MP3's). Again we find the compressability not surprising. The graph is

a shocker, though. Looking closer we see that, with the exception of the very beginning, this does, indeed, look like the graphs of MP3's we have seen so far. There is no indication in the raw data what the huge spike at the very beginning might be -- just a lot of 0xFF's. This is reflected in the run lengths in that it has two outliers not pictured in the graph at 2027 bits and 3066 bits. The run lengths graph for this MP3 looks very much like the one for the first MP3. There is a higher proportion of zero runs in the 20 bit to 50 bit range, and it is uninterrupted. The runs of ones once again exhibit the miniature spike and decline at about 80 bits, though not as markedly as in the first graph.

FTP command session #6
Size: 603 bytes
Normalized number of runs: 0.518657
Result of universal statistical test: 0.752321

FTP data session #6
Size: 2589676 bytes
Normalized number of runs: 0.487517
Result of universal statistical test: 9.061785

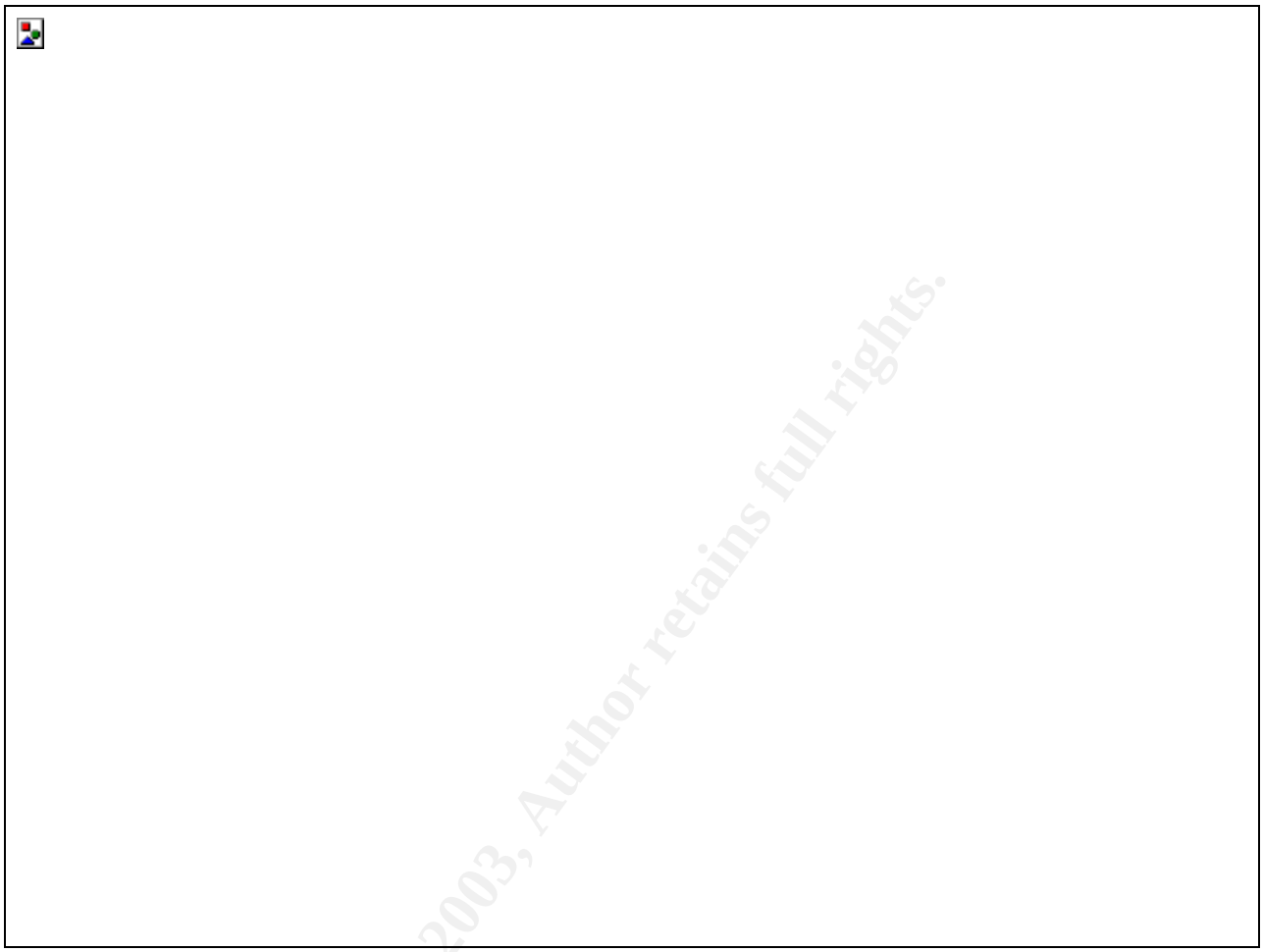
Here we have two HTML documents transported over HTTP. Very compressable with a tendency to lots of zeros, and a fairly wide range in the block frequency test characterize this traffic. The run lengths have less of a bump around 6 bits, but they nonetheless look very similar to the FTP command channels that we saw in the previous six graphs. Not surprising. Not suprising also, that the longest runs in the FTP command channels and these HTML documents are between 7 and 9 bits. After all, all of those data are text, and text is primarily in the middle low range of the values that a byte can take on. In fact, it is very likely that we would seldom see a byte in ASCII text that uses the most significant bit, meaning that, at the latest, we will have a zero every byte, which makes a long run of ones impossible (again, on the assumption that extended ASCII characters are not used). The only way, then, that we could have a long run of zeros is if a byte (or more) is the null byte, which is also very unlikely in normal text. The run can, of course, also cross byte boundaries, but we still would not expect to see long runs of zeros.

The next example is nearly identical, and even comes from the same set of Web pages (the POV-Ray Web site). It will not be commented on separately.

HTML #1
Size: 5449 bytes
Normalized number of runs: 0.493370
Result of universal statistical test: 2.455436

HTML #2
Size: 5471 bytes
Normalized number of runs: 0.493808

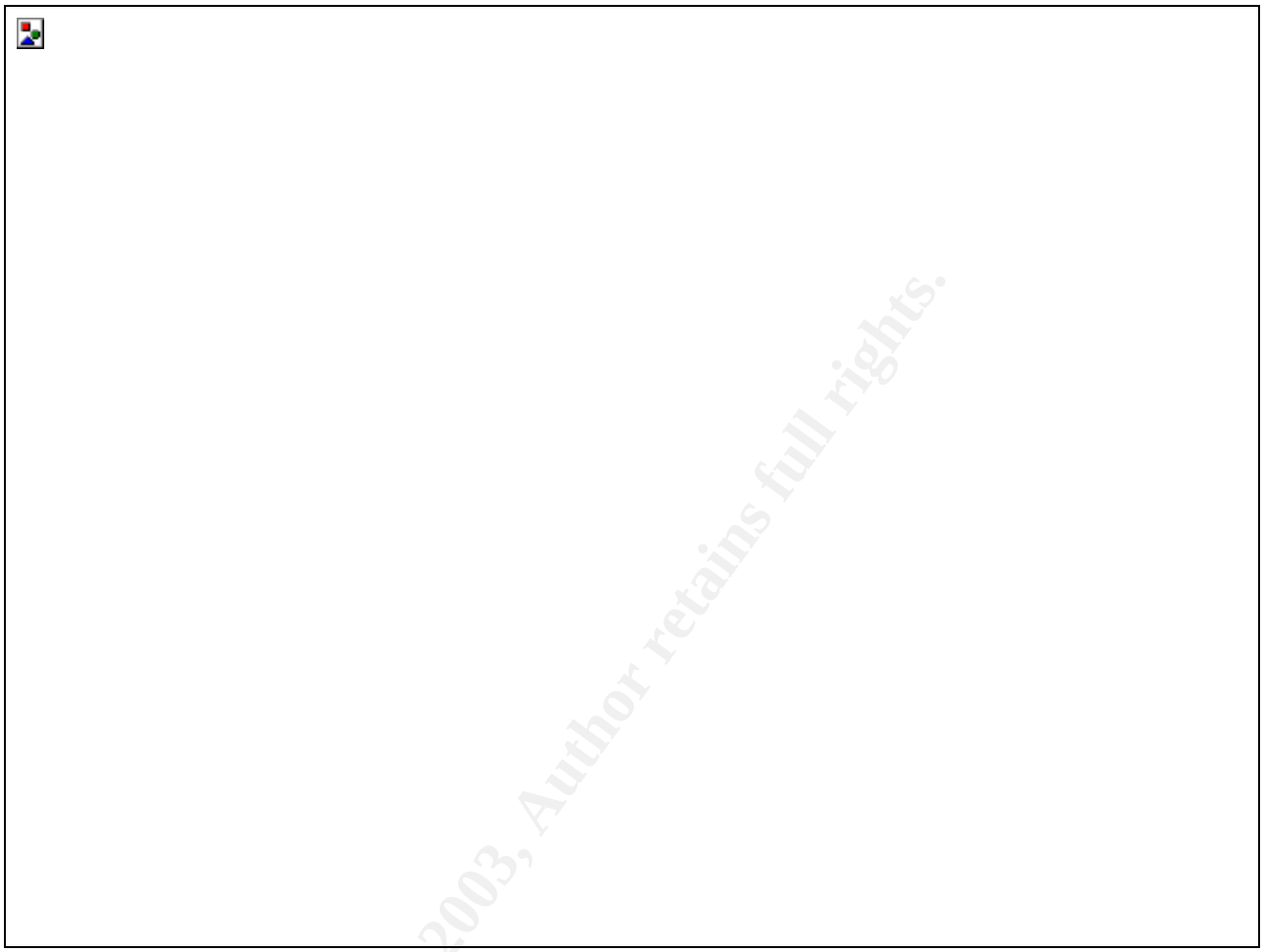
Result of universal statistical test: 2.453681



This is once again an HTML document over HTTP, albeit a much larger one. This is the documentation for gnuplot. Here we see nothing more than a slight exaggeration of the qualities we have already identified. It looks a little less compressable, there is a stronger tendency towards zeros, and the negative range for the block frequency test is a bit larger, though most of the blocks still yield frequencies between -0.1 and 0. The bump of zero runs around 6 bits is stronger. A quick visual inspection of the HTML documents seen thus far indicates that the POV-Ray Web pages were more likely created with assistance from a Web page editor, while the gnuplot documentation looks like handwritten HTML.

HTML #3
Size: 395058 bytes
Normalized number of runs: 0.485447

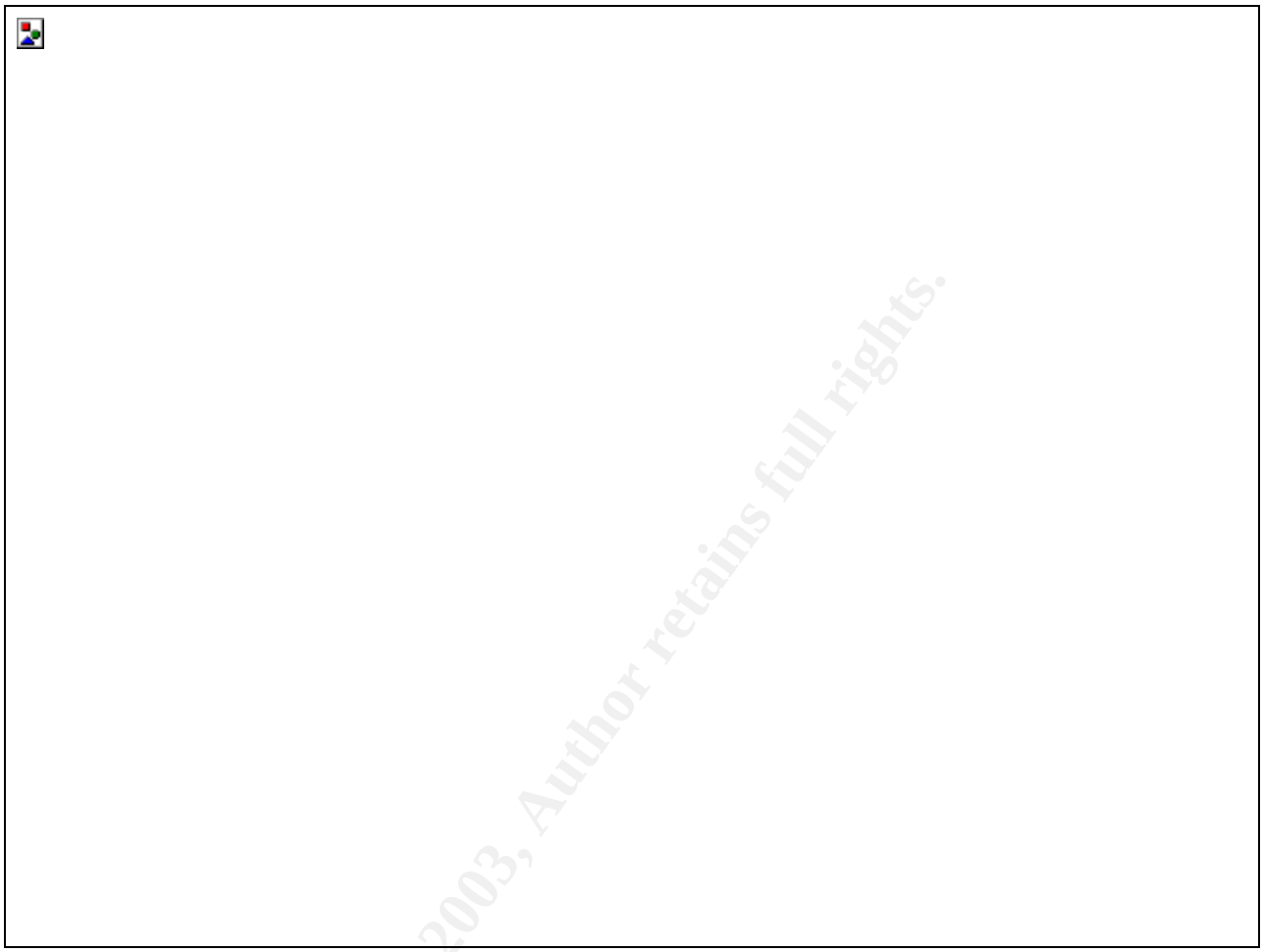
Result of universal statistical test: 4.173580



One more HTML document is called for to make certain that a variety of HTML pages all look about the same. This is the homepage for OpenBSD. This is a good time to mention that the runs of ones in text transfers has not once gone over 6 bits in length.

HTML #4
Size: 9556 bytes
Normalized number of runs: 0.498522

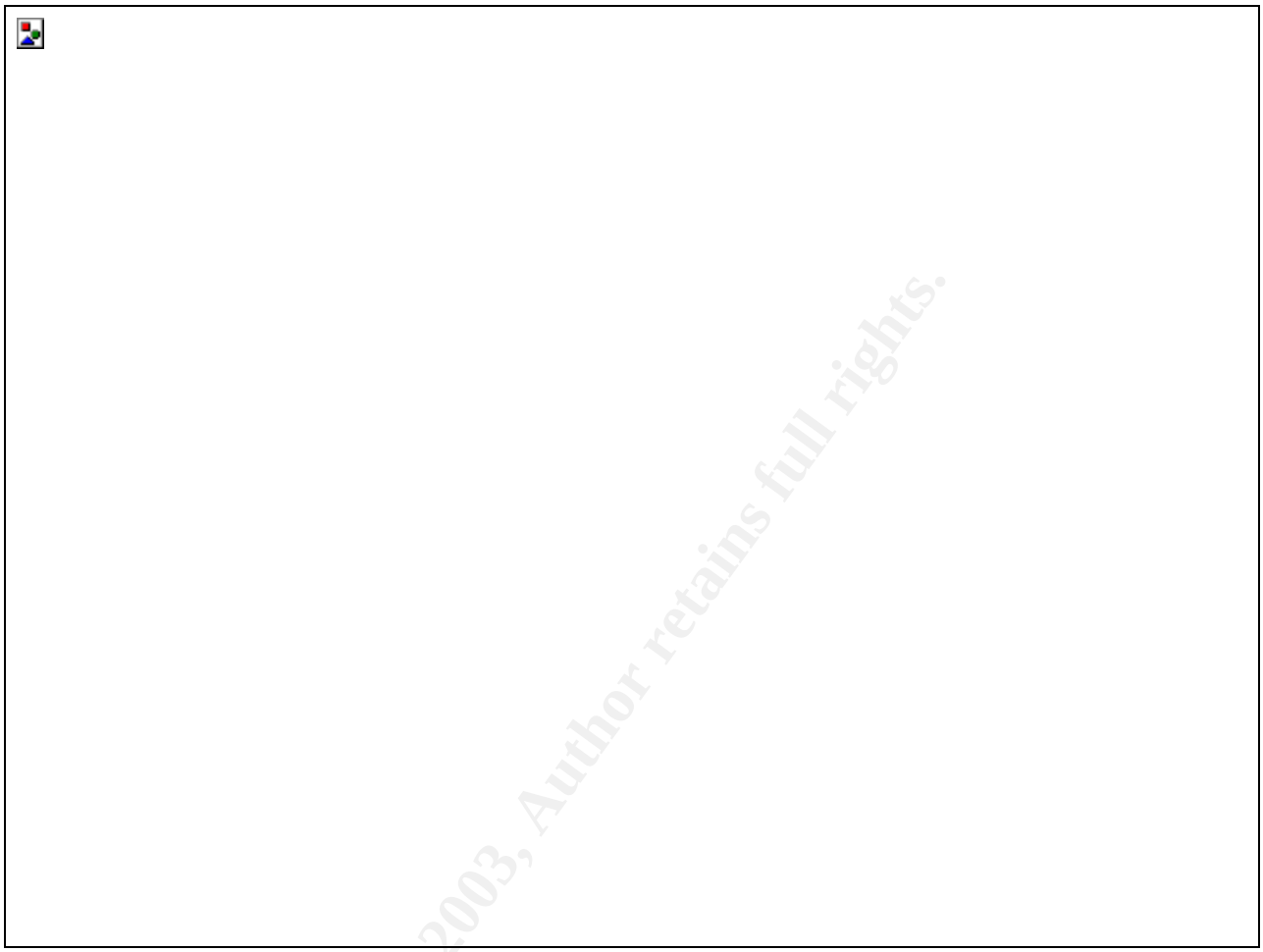
Result of universal statistical test: 2.976634



The next test is a slightly different beast. In this case, the data are still being sent over HTTP, and it is a Web page, but the server has taken the client up on the offer to compress the data before sending them. The compression is once again gzip. The low result for Maurer's test is not what we would expect, but may end up reflecting HTTP to some measure. We also see HTTP in the block frequency graph as the blocks up to about 100 that have more of a tendency towards zero than the following blocks. The compressed blocks look much like the compressed file in the second FTP transfer from above in terms of balance above and below the axis and range above and below the axis. The runs once again look like the compressed file transfers we saw above, as they should. The run of 52 zeros is a bit odd, but not critical. The page is from MySQL AB's Web site.

Compressed HTML
Size: 7289 bytes
Normalized number of runs: 0.500806

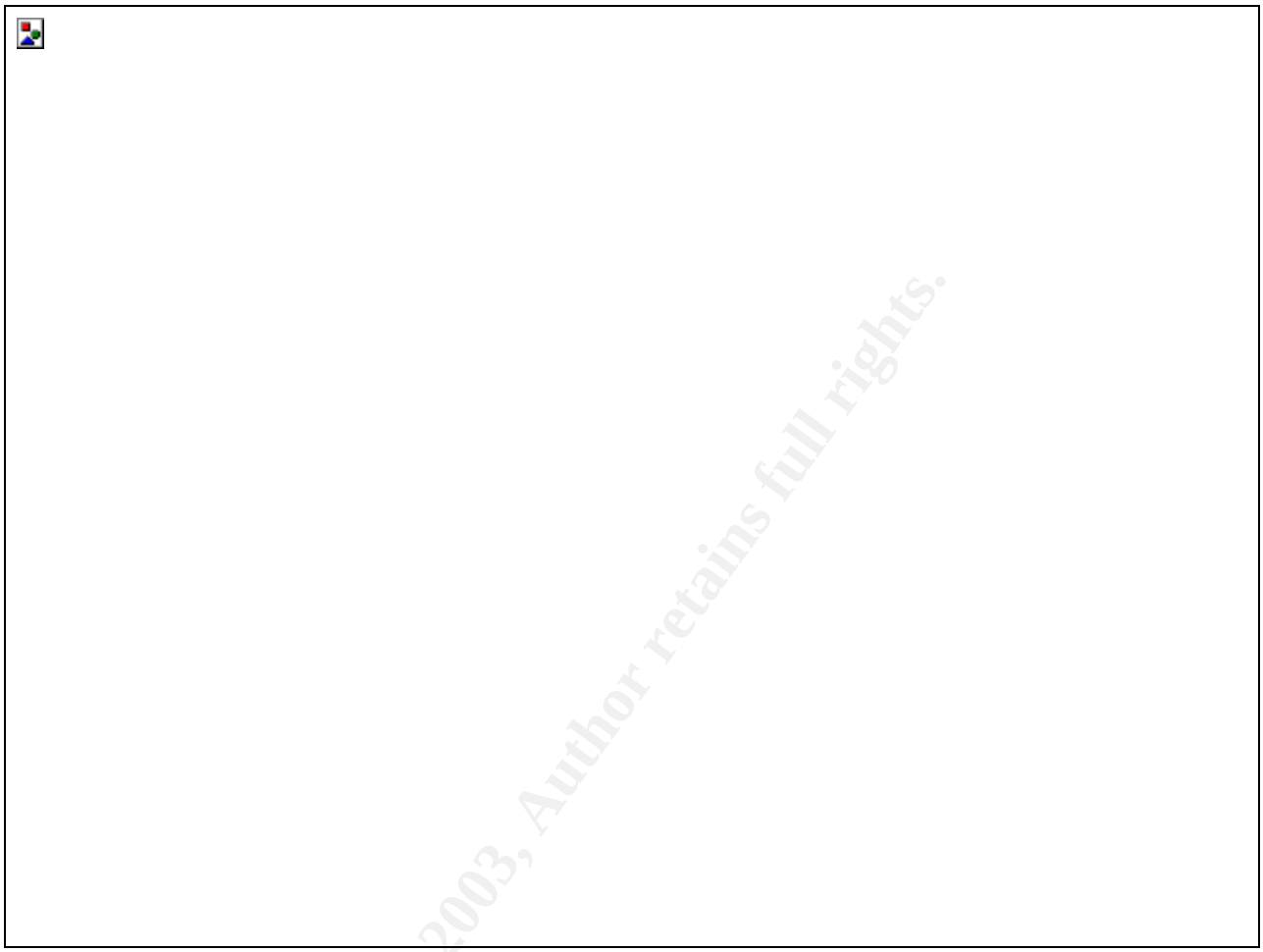
Result of universal statistical test: 2.403431



The next test shows us the result of the "Connection: Keep-Alive" HTTP header. Here we have two HTML documents followed by a JPEG image. The boundary between HTML and JPEG is clear in the block frequency graph. While a slight peak is observable around 6 bit zero runs, the text is overpowered by the JPEG image in the graph of run lengths. For all practical purposes, we can consider this graph to reflect the run length properties of a JPEG image. These data are from SETI@Home's Web site.

HTML and JPEG over HTTP
Size: 14881 bytes
Normalized number of runs: 0.495934

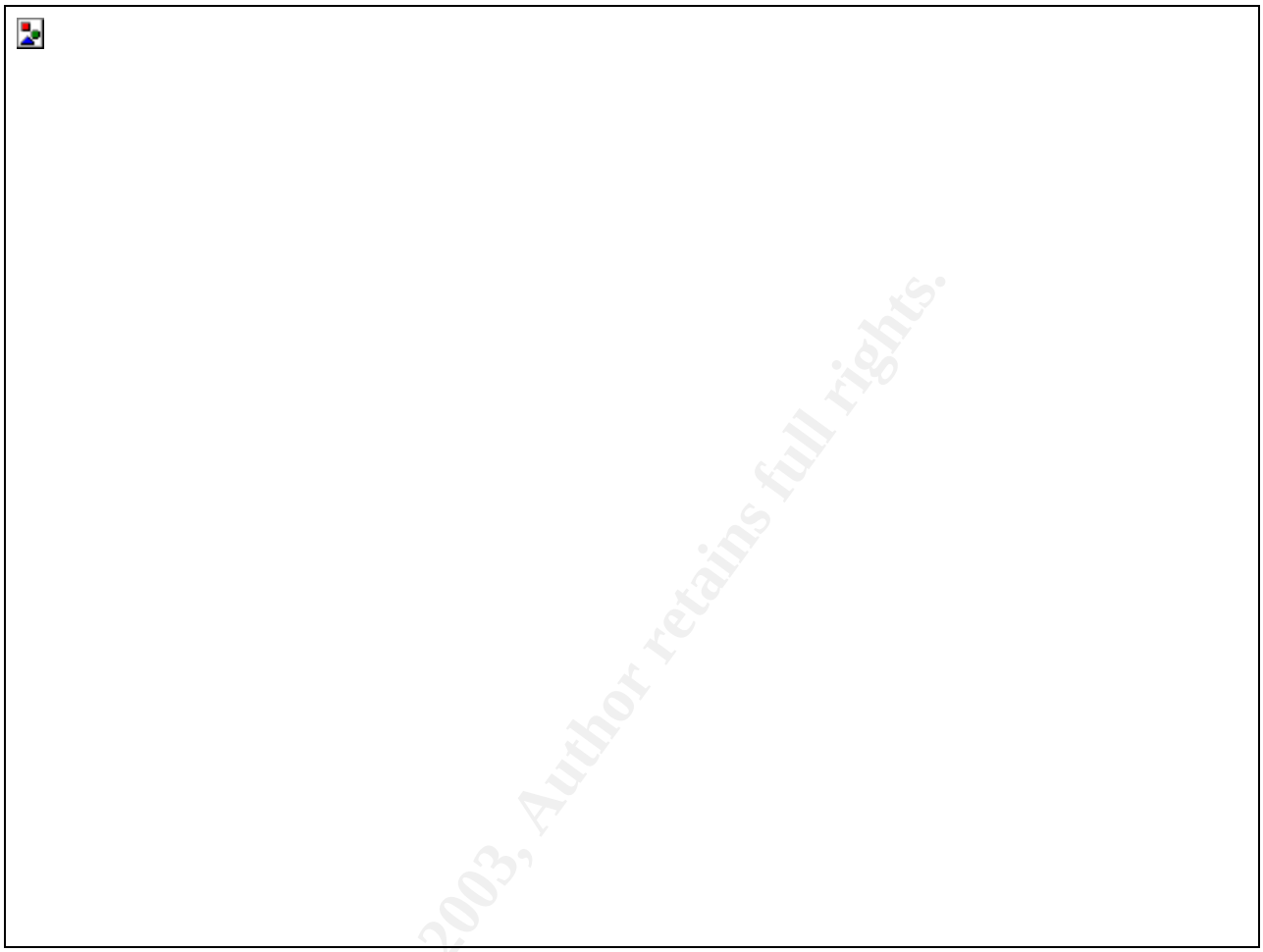
Result of universal statistical test: 3.077624



By now we can guess what the next test is. Relatively low compressability along with a huge negative dip in the block frequency graph at the beginning and a fairly even distribution after that ... looks like a .exe, and it is. It is the Windows graphical client for Folding@Home. The run length graph in its present form is all but useless, since we can't see the first one hundred or so bars. What is interesting are the huge runs of zeros. These reflect the header that is visible in the block frequency graph.

.EXE over HTTP
Size: 331667 bytes
Normalized number of runs: 0.473180

Result of universal statistical test: 6.714121



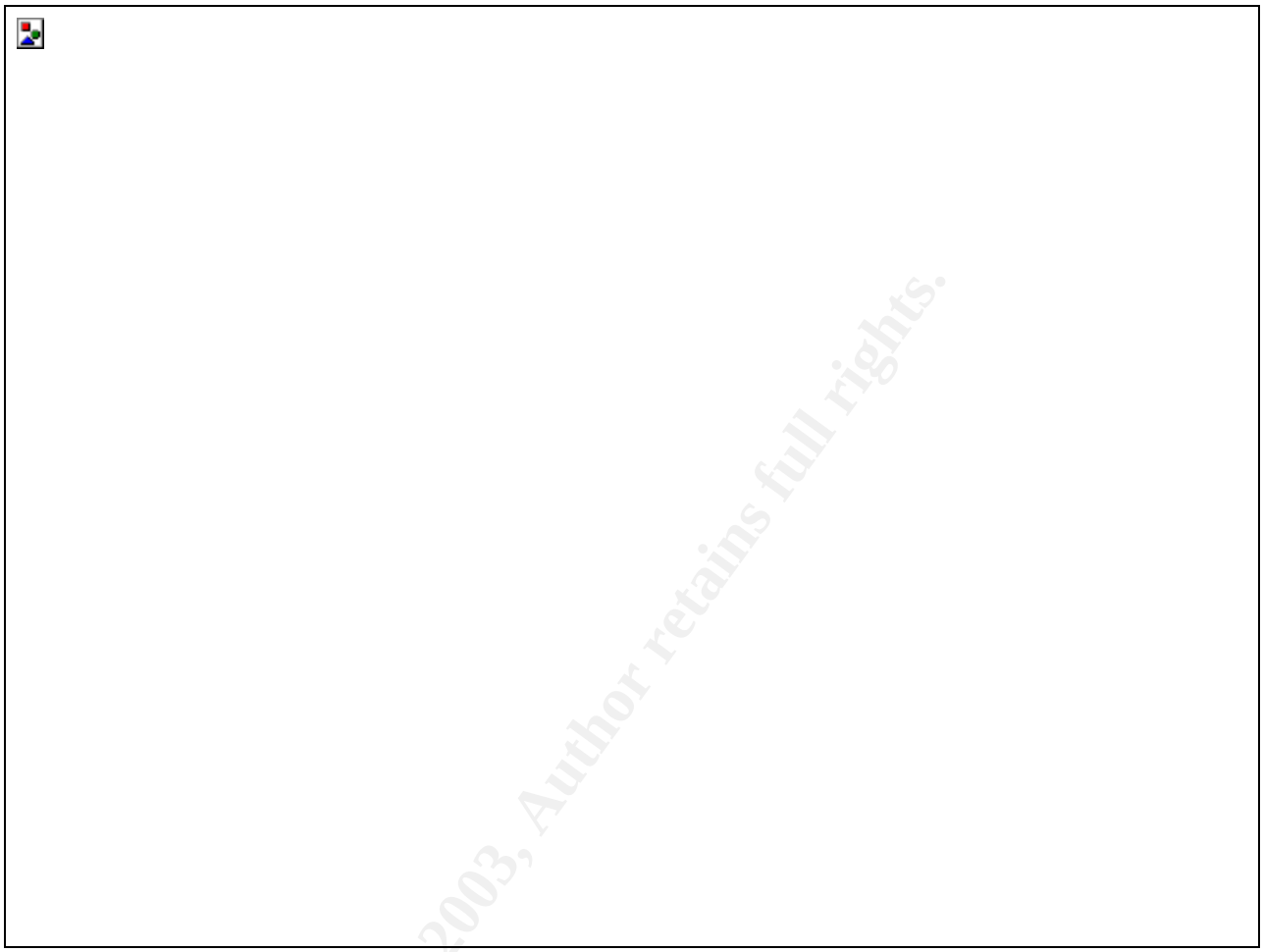
This test is of a JPEG image transferred over HTTP. One can see the pure HTTP at the beginning of the graph, followed by what is probably a header in the JPEG image that is primarily negative. The middle compressability is expected, since JPEG includes compression. What is unexpected is the fact that the blocks in the block frequency test are unwaveringly positive. This does not match with the JPEG image we saw earlier mixed with the HTML documents, nor with the next test results. Because this is so curious, the image was tested with stegdetect 0.5 (Provos, "Steganography ... Stegdetect") to see if there are hidden data in the image. That test came up negative, so we assume the image is clean. The run lengths do not follow quite as clean a linear descent to zero as the gziped data we have seen, but these are still obviously well compressed data. A close look indicates that there is a much higher proportion of runs of zeros of length 1 than of runs of ones of length 1. After that, the runs of zeros are significantly less than those of ones up until runs of length 16 bits. The dominance of zeros in the longer lengths is probably attributable to the header in the JPEG image. All that is to say that the run lengths graph supports the result in the block frequency graph. This image is from the POV-Ray hall of fame.

JPEG over HTTP #1
Size: 136252 bytes
Normalized number of runs: 0.483756
Result of universal statistical test: 6.053724

This image is also from the POV-Ray hall of fame, and much more closely matches what we saw earlier in the mixed transfer. It is a much more evenly distributed block frequency graph across the axis. The compressability is a bit higher. Here, too, we can see the HTTP headers at the beginning of the graph. What is missing that we saw in the first JPEG image in the mixed transfer is the large negative dip. This image also has a much smaller range. The run lengths graph holds no surprises.

JPEG over HTTP #2
Size: 268124 bytes
Normalized number of runs: 0.510463

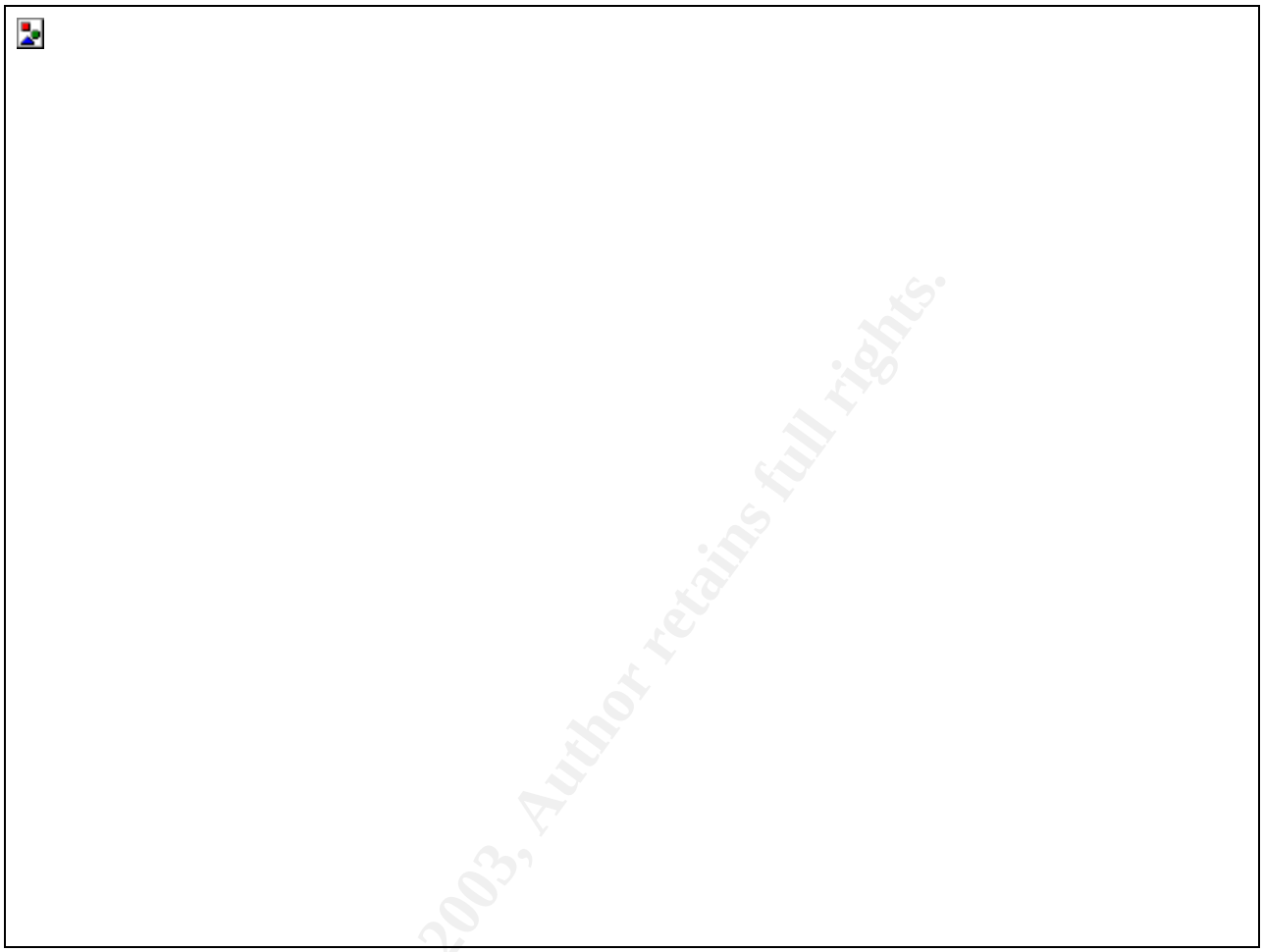
Result of universal statistical test: 7.147094



Analyzing the next data transfer is easy. This is an SSH data transfer. Knowing that SSH encrypts with highly regarded publicly available encryption algorithms (probably Blowfish in this case, based on the author's configuration files), it is no surprise that the compressability is very low, the blocks in the block frequency test are very evenly distributed, and that the range of same is small with very little in the way of spikes. Not surprising, also, that the normalized number of runs is closer to 0.5 than the other tests. The run lengths graph displays perfect properties of randomness for about the first 25 lengths. The oddballs after that are likely due to the negotiation phase. The negotiation phase of the SSH connection is visible at the very beginning of the graph.

SSH file transfer
Size: 4539007 bytes
Normalized number of runs: 0.500119

Result of universal statistical test: 10.171163



We have seen a lot of normal traffic, but now it's time to examine some abnormal traffic to see if that differs markedly from the normal traffic. If not, the technique is useless.

The first graph of abnormal traffic we see is a series of four CodeRed I attacks against a patched IIS Web server. Here we see only the communication from client to server, because this shows us what is, in this case, more interesting. The first three graphs are identical for both types of graphs, but the fourth graph is somewhat different. In the case of the fourth graph, there was no TCP handshake, and the attacking host seems to have sent the entire set of packets without any kind of a response. The sending host seems to be very confused, because it also sends a FIN later, then another series of five or six packets. That, though, is not the focus of this analysis. The astonishing part of the results for these packets is the normalized number of runs. This is the first time we have seen them when they are not clustered around 0.5. Besides that, all of these graphs are visually distiguishable from the previous HTTP graphs. If nothing else, the last few blocks in the block frequency graph that are far below the axis and all equally sized are a dead giveaway that something isn't right, as is the fact that we would expect this to be primarily text, but we see many, many runs longer than 9 bits. Not forgetting that this is

only the client to the server, we would expect this part to look something similar to the FTP command channel communication or HTML files from above assuming the GET or even HEAD HTTP methods (but not necessarily with PUT) are used.

CodeRed I attempt #1
Size: 4039 bytes
Normalized number of runs: 0.378590
Result of universal statistical test: 2.035687

CodeRed I attempt #2
Size: 4039 bytes
Normalized number of runs: 0.378652
Result of universal statistical test: 2.035875

CodeRed I attempt #3
Size: 4039 bytes
Normalized number of runs: 0.378528
Result of universal statistical test: 2.035663

CodeRed I attempt #4
Size: 2575 bytes
Normalized number of runs: 0.375534
Result of universal statistical test: 1.276910

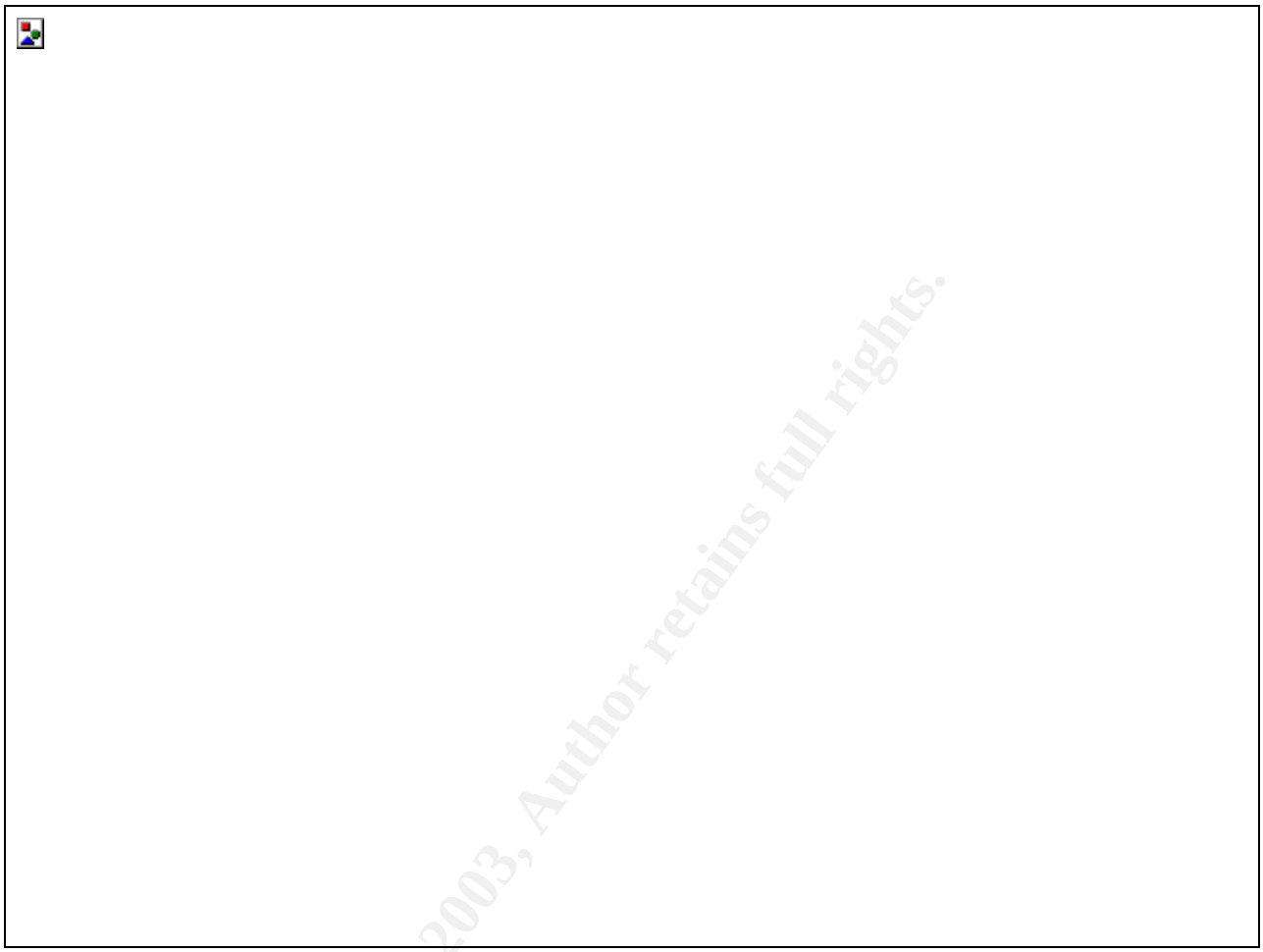
Now we'll take a look at a hacked SSH session. This is a session to OpenSSH 3.2.3 using GOBBLES' exploit. (Duke) The commands run on the hacked box were:

```
cd /usr/share/man
find . -name \*.0 -exec cat \{\} \;
```

These commands were executed for the sake of being able to see the difference between a normal SSH session and a broken one. The very beginning of the session actually looks like a normal session, because it **is** a normal session up until the point where the overflow is executed. This part of the session is so small in comparison to the output of the above commands, that it isn't visible in the graph.

Hacked SSH session
Size: 12779456 bytes
Normalized number of runs: 0.451643

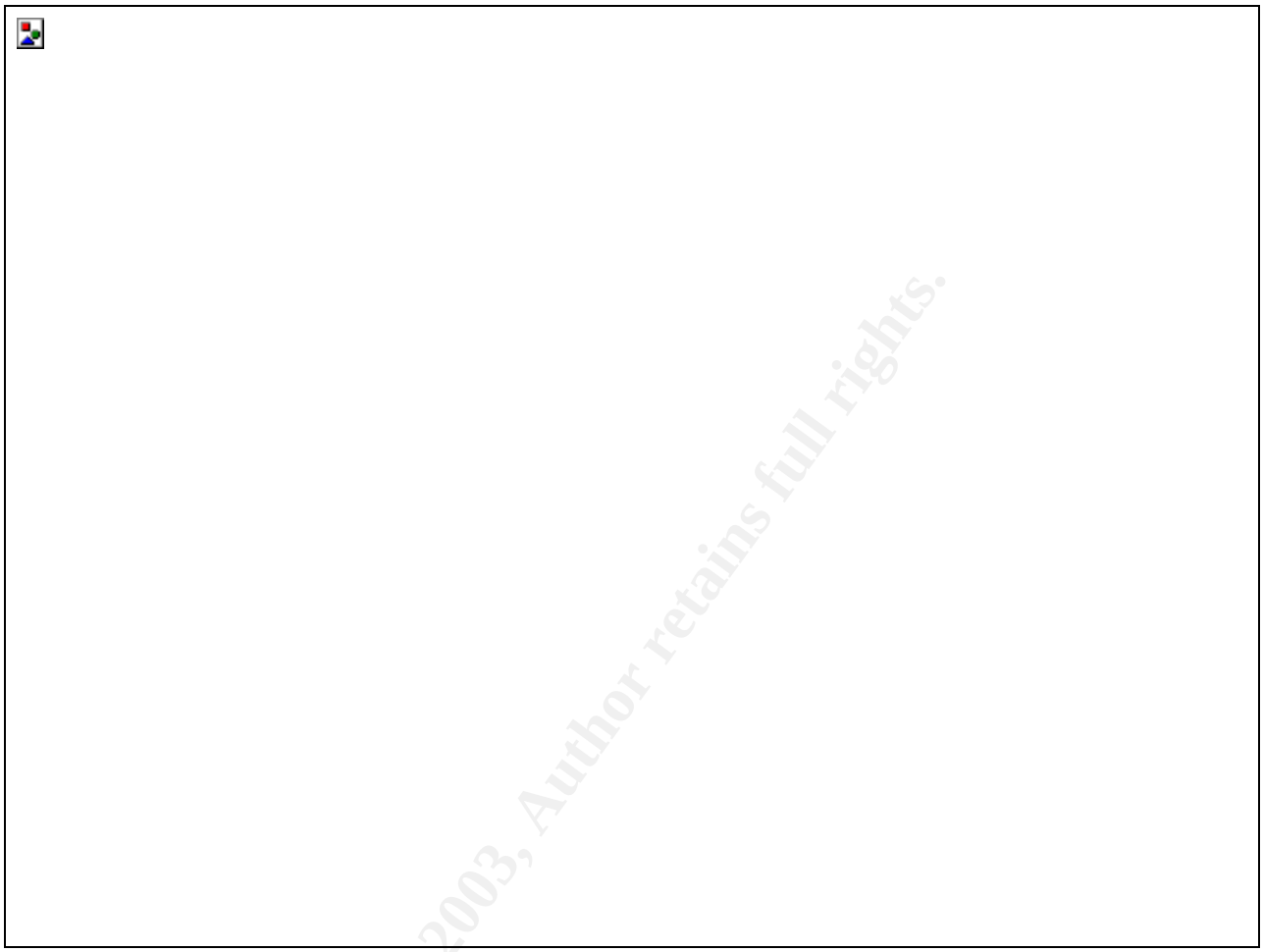
Result of universal statistical test: 6.340763



Since we examined normal FTP traffic above (although mostly for the purposes of seeing how well various kinds of data can be profiled), we present one last attempted exploit. This is an exploit for the BisonWare FTP server penned by Rose Labs. (Vampiro) It is being run (slightly modified) against the FTP server that comes with OpenBSD 3.2. The exploit makes use of a buffer overflow in the USER (actually LOGIN in the original code) and PASS FTP commands. The differences in these graphs are impossible to miss.

FTP exploit
Size: 1798 bytes
Normalized number of runs: 0.501390

Result of universal statistical test: 1.409577



The one test parameter that was almost completely ignored in the analysis was the normalized number of runs. The reason for that is that the statistic turns out to be useless. The FTP transfers seem to show a pattern of the FTP protocol (the ASCII text) having a normalized number of runs between 0.5 and 0.52, while the binary data in the data channel have runs between 0.48 and 0.5, both roughly speaking and on average. We would then expect to see numbers similar to the FTP command channel in for the HTML over HTTP transfers. We don't. Instead, we see numbers more in the range of the FTP data channel for HTML. Going the other way, the results for the second JPEG image show the runs significantly (in terms relative to the surrounding values) above 0.5, which was until then primarily the domain of text. The only time we see values for this test statistic not in the very narrow range of about 0.45 to 0.55 is in the CodeRed attacks. Still, this number does not seem to give us any good way of catagorizing data.

It is astonishing to see the wide variety of statistical properties that data of the same sort (JPEG image or gziped data) can exhibit. The JPEG over HTTP #1 and HTML #3 graphs both exhibit extreme tendencies not found in other block frequency graphs of similar data. One explanation is the difference in the block sizes of the different graphs. HTML #3 is a profile for a big document, and it could be that the tendency towards zeros that we

identified in the other HTML documents is showing up here as being almost exclusively zero because the block size is larger. This idea does not hold up for two reasons. One would expect to see bars closer to zero in the graph for HTML #3 if the resolution were a problem, since we are really talking about something akin to averaging out the graph. The two JPEG images over HTTP also disprove this idea, because the graph that stays exclusively above the axis is the graph for the smaller of the two files.

The run lengths test, although it also exhibits a wider berth in statistical properties for the same kinds of data than we might like, is still relatively stable and provides a good check against the block frequency test. That is, those two tests together could do a half decent job of identifying a wider range of network traffic (and labeling it safe or dangerous) than either test alone.

Despite that, this property of wide statistical variety makes the use of this technique difficult to apply to some network communication. HTTP and SMTP, for example, would be nearly impossible to analyze, because everything is transported over those two protocols. SSH, on the other hand, would be a snap, because it fits very predictable patterns. One-sided traffic might also lend itself better to analysis, even with difficult protocols like HTTP and SMTP. We have already seen that analyzing only the client side of a CodeRed attack turns up a signature that is likely to be identifiable in the wild as bad traffic without knowing about it ahead of time.

Now that we have an idea of how the technique works and what it is capable of, it is time to discuss the practical issues. The first practical issue that was mentioned before the analysis is speed. These tests are very slow. Even using the highest optimization the C compiler makes available, and running on a 1.47 GHz AMD processor these tests were about 31 times slower than they would need to be to keep up with 100Mbit/s traffic. Sounds terrible, but there is still hope. We already identified the runs test to be of little value, so we can remove that. Maurer's test is interesting and truly useful in characterizing the traffic to be analyzed, but we still might be able to live without it if we can use some other parameters in helping us identify the traffic. The run lengths test does a good job of covering Maurer's test, because it can also give us an indication of how compressable the data are. Elementary (and probably not very accurate) profiling of the code used to generate these statistics reveals that Maurer's universal statistical test takes about 54% of the time that the application runs, the run lengths test takes 26%, of which more than half is simply the use of the linked list implementation for keeping track of the number of occurrences of each run length, and the number of runs test takes an additional 10%, as does the block frequency test. That means that if we limit ourselves to the block frequency and the run lengths tests we get about a 3x speed up. Next we replace the linked list implementation in the run lengths test with an array implementation, much like NIST did, only a bigger array (NIST used a maximum of seven entries, and we would want more along the lines of 100 to 200). Even assuming thousands of connections and a 64 bit machine word, this implementation would only take a few megabytes. That time/space tradeoff effectively cuts the time to maintain the runs list to nothing. This leaves us needing about a 7x speed up. With other tricks or limitations, like the ones discussed above (the client side of normal Web communication is probably much less

than seven times that of the server side), that may be attainable, although it is still much higher than we would like.

Although we have concentrated on real time analysis, this technique lends itself very well to offline analysis with an IDS like Shadow. If the link being watched averages about 10Mbit/s saturation or less, an offline analysis of an hour's worth of data would take less than an hour to process. Otherwise, the analyst could first extract the data to be analyzed with this technique.

The next question we need to investigate is how we might expect to see this used in a real intrusion detection system. Every network is different, as is the traffic in every network. Even if we want to do something as simple-sounding as using this in a protected public network segment with Web servers to identify new attacks, we would still need a way of telling the IDS what kinds of traffic this technique should consider to be normal. In short, a profiling period would have to take place. In this period the IDS would create something of a database to tell it later, which characteristics of the traffic it is seeing are normal. Administrator interaction may be necessary for this process. The parameters should definitely be tunable by the administrator to avoid many false alarms in case the profiling creates a profile that is too "tight", or that is skewed.

One more question remains, and that is what traffic would benefit from this kind of analysis? We have already identified that the process is slow and that identification is hard in some cases, so it makes sense to try to use it for certain types of traffic and not for others. An obvious choice is to evaluate SSH and IPsec traffic. This traffic is very easy to identify, and it would be equally easy to find outliers. Another very useful place to use this technique may be in front of Web servers. Web servers are constant targets for new attacks, and they are also often high profile machines. It would be wonderful to be able to identify new attacks against Web servers before they are widely known and a signature is written for IDS's. The practicability of this suggestion would vary greatly from site to site. We would want to evaluate the client side of the connection only in order to save CPU clock cycles and remove data that would confuse this technique, but even then this may not be an option at large sites whose Web servers have thousands of hits a second. Other operations with a few Web servers at the most, a small to medium Internet connection, and not much traffic may be able to do this without an oversized IDS machine in the protected network. Even under these restricted conditions we can detect only a limited class of attacks. Attacks that statistically look like normal traffic will not be picked up. That means, an attack against a Web server that relies on a certain encoding of the URL and does nothing more than cat /etc/passwd would probably not be picked up, because the attack is still more or less text. A buffer overflow attack might be identified, unless it were padded with an excerpt from one of Shakespeare's plays (for example).

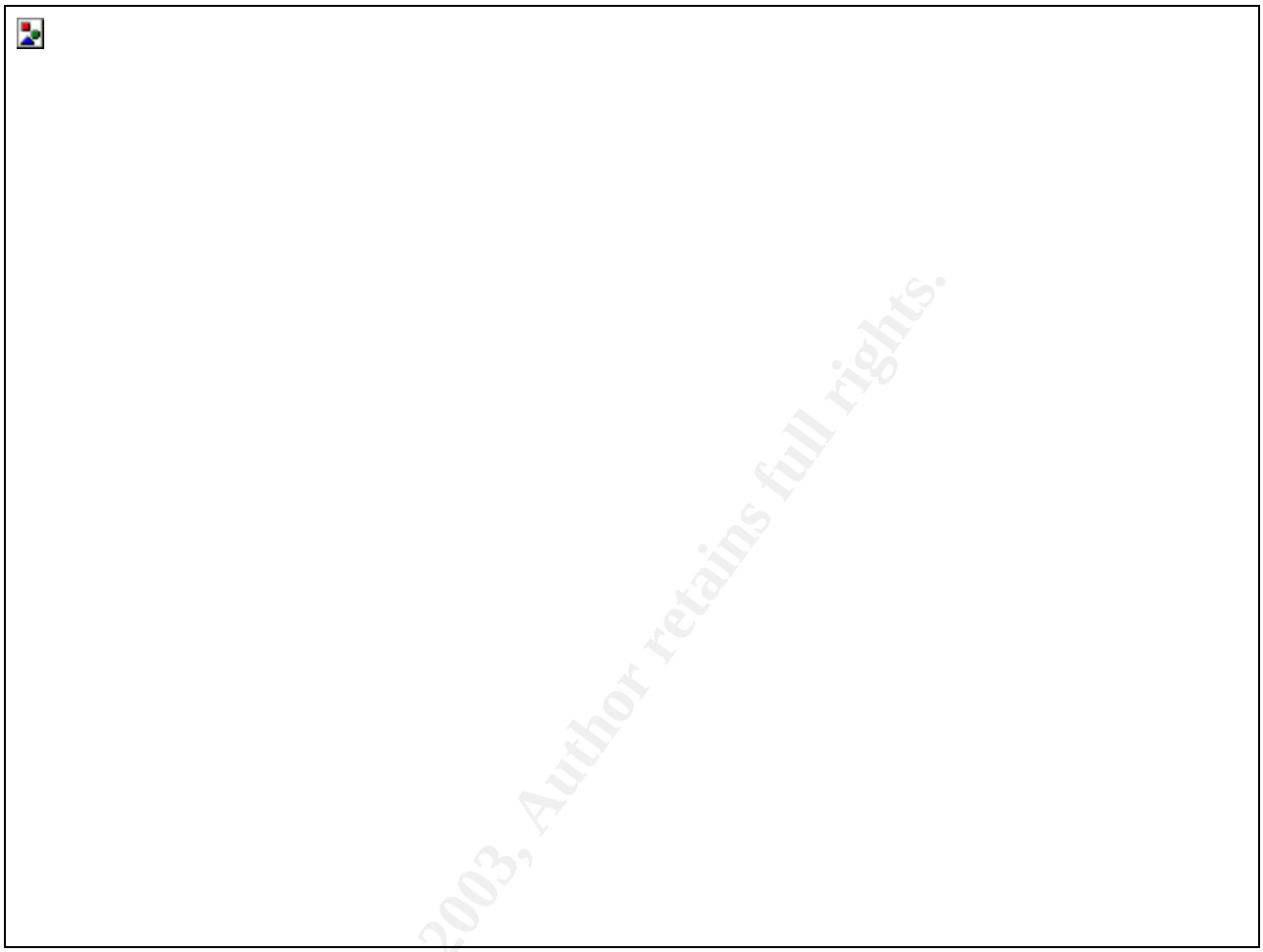
Another idea, forwarded by Jamie French in a private communication (French) is to try to detect covert channels. We will investigate that option here, first with a practical example using the data we have collected thus far, then from a theoretical point of view. We have already seen two results for JPEG images over HTTP. One of them was so odd that we were prompted to test for the presence of steganographically hidden data using Niels

Provos's Stegdetect. The results came up negative. Now we would like to see what those results look like when we know that there are hidden data in the image. We will be using the complementary tool from Niels Provos to hide data in JPEG images: OutGuess. OutGuess does its best to hide a small enough quantity of data in as sneaky a way possible, making it difficult to detect. The Web site claims that OutGuess "[p]reserves statistical properties of the cover medium, no known statistical tests based on frequency counts can detect steganographic content. Determines the size of a message that can be hidden safely." (Provos, "OutGuess - Download Page") Using OutGuess's suggestion, we were able to hide 5500 bytes in the first image and 7000 bytes in the second image. The results are shown in the two graphs below.

JPEG with steganographically hidden data #1
Size: 89312 bytes
Normalized number of runs: 0.492703
Result of universal statistical test: 5.212688

JPEG with steganographically hidden data #2
Size: 268124 bytes
Normalized number of runs: 0.510463

Result of universal statistical test: 7.147094



Comparing these results to the previous results shows us that the statistical properties are very different. OutGuess had difficulty hiding data in the second image, and was apparently not able to do it using the hiding techniques it employs. Thus, we would expect the altered second image to send up alarms under statistical tests. The first image provided OutGuess with no problems. The images have swapped statistical properties; originally the first image was primarily positive and the second image showed a good balance across the axis, but now it's just the opposite. The other test statistics are not drastically changed. In fact, the normalized number of runs and the results of Maurer's test for the second image are nearly identical. The first image has a somewhat different result for Maurer's test, and the size is very different. Presumably the changing of a few bits made the image much more compressable. Naturally, both of the images with hidden data look identical to the "clean" images with the naked eye. The point here, though, is that, while the statistical tests do indeed return different results for "loaded" images, the results do not look all that different from the results for different clean images. That means that this technique in its present form would not be suitable for detecting covert channels of this kind.

From the theoretical perspective, this technique is simply not sensitive enough to pick up on steganographically hidden information. It is designed to analyze large amounts of data and find traffic that lies far outside of the norm, hopefully without crying wolf too many times. As Gina Fisk rightly points out in her research presentation on preventing steganography in Internet traffic:

500 million network packets leave an average large site each work day. Modifying just one bit on each packet would result in a loss of over 26 GB of data annually. TCP, IP, and UDP headers provide embedding opportunities for more than 8 bytes per packet header (i.e., 4 GB per workday) (Fisk)

Steganography at levels such as these will not be detectable with rough statistical tests such as the ones presented in this paper. The only possibility for that would be if the original data are so packed with hidden data that the statistical properties are drastically changed. Even then, these tests would not be able to identify the cause of the difference, and the administrator would have to know what (s)he was looking at. A better solution is the one proposed by Gina Fisk in the above mentioned presentation, which makes for good reading. The solution is active wardens.

Since we have mentioned them, we now must dedicate a small paragraph to explaining what active wardens are. Active wardens normalize all traffic at all layers to remove possible covert channels. At the IP layer, they might zero out the TOS bits, at the TCP/UDP layer they might change the source port, and at the application layer they might do something like normalize HTML by removing all comments, ordering the tag options in a standard fashion, changing all HTML markups to lowercase, and so on. (Fisk) The question of how much promise this technology holds is still open. To this author, it sounds really great on paper, but it will not likely be very implementable, and will probably be prohibitively expensive to use if it is ever implemented.

A small technical problem needs to be addressed, and that is the issue of turning off certain alerts from this mechanism. It is hard for this method to distinguish between Code Red and a brand new buffer overflow. It would be irritating to receive an alert for every Code Red attempt against a half dozen Web servers if we know they are already patched and we turned off the alert for Code Red in the signatures for our IDS. A method of correlation between the two identification mechanisms would have to be developed. It may be as simple as using this technique after every packet has traversed the IDS signatures and only if an alert was not generated by the signatures.

Conclusion

The idea of using randomness tests as a way to find as yet unknown attacks is not a bad one. The hypothesis presented in the introduction has, under certain conditions, been verified. The idea needs to be refined both on a theoretical level and on an implementation level to find a more workable and faster set of tests and the best place to apply the technique. The great promise of this technique may well be realizable: identification of zero-day exploits. Further, more research needs to be done into how

applicable these tests are to the layer three and four headers of network traffic, which is a problem the author was not able to tackle due to time constraints. Although the author said early in this paper that this is best handled with signatures, that is mostly true for old, well-known protocols. The advent of IPv6 may bring new attacks at the network layer. Other statistical tests may be necessary for these data due to the size of the data. This, of course, assuming that active wardens are still beyond the horizon.

Bibliography

Computer Security Division, NIST and Statistical Engineering Division, NIST. "rng0". Random Number Generation and Testing. 8 December 2000. National Institute of Standards and Technology. 23 February 2003. <http://csrc.nist.gov/rng/>.

Duke (pseudo.?). GOBBLES Security. Immunity Security. 23 February 2003. <http://www.immunitysec.com/GOBBLES/exploits/sshutup-theo.tar.gz>.

Fisk, Gina. "Eliminating Steganography from Internet Traffic with Active Wardens". 27 November 2002. University of Southern California. 23 February 2003. <http://netweb.usc.edu/cs551/slides/gina.ppt>.

French, Jamie. "RE: Possible topic and outline for GCIA v3.3 practical". E-mail to Andrew Jones. 21 January 2003.

Provos, Niels. "OutGuess - Download Page". OutGuess. 6 March 2002. 23 February 2003. <http://www.outguess.org/download.php>.

Provos, Niels. "Steganography Detection with Stegdetect". OutGuess. 26 January 2002. 23 February 2003, <http://www.outguess.org/detection.php>.

Rukhin, Andrew et al. "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications: NIST Special Publication 800-22 (with revisions dated May 15, 2001)". Random Number Generation and Testing. 15 May 2001. National Institute of Standards and Technology. 23 February 2003. <http://csrc.nist.gov/rng/SP800-22b.pdf>.

Schneier, Bruce. Applied Cryptography: Second Edition. New York, New York: John Wiley & Sons, Inc., 1996.

Vampiro, Conde. 29 February 2000. Packet Storm. 23 February 2003. <http://packetstormsecurity.org/0003-exploits/RLbison.tgz>.

"Steganography Software - Unix, FreeBSD, Linux". StegoArchive.com. tripod. 23 February 2003. <http://members.tripod.com/steganography/stego/softwareunix.html>.

Appendix: Source code for randomness tests

The following is the source code that was modified from the original NIST source code from the statistical randomness test suite. A couple of header files are not included because they are unchanged. The reader is directed to NIST if those files are of interest. The URL is in the references above.

arjassess.c:

```
/* ---------------------------------------------------------------------
------
   Title       :  The NIST Statistical Test Suite

   Date        :  December 1999

   Programmer  :  Juan Soto

   Summary     :  For use in the evaluation of the randomness of
bitstreams
                  produced by cryptographic random number generators.

   Package     :  Version 1.0

   Copyright   :  (c) 1999 by the National Institute Of Standards &
Technology

   History     :  Version 1.0 by J. Soto, October 1999
                  Revised by J. Soto, November 1999

   Keywords    :  Pseudorandom Number Generator (PRNG), Randomness,
Statistical
                  Tests, Complementary Error functions, Incomplete
Gamma
                  Function, Random Walks, Rank, Fast Fourier Transform,
                  Template, Cryptographically Secure PRNG (CSPRNG),
                  Approximate Entropy (ApEn), Secure Hash Algorithm
(SHA-1),
                  Blum-Blum-Shub (BBS) CSPRNG, Micali-Schnorr (MS)
CSPRNG,

   Source      :  David Banks, Elaine Barker, James Dray, Allen
Heckert,
                  Stefan Leigh, Mark Levenson, James Nechvatal, Andrew
Rukhin,
                  Miles Smid, Juan Soto, Mark Vangel, and San Vo.

   Technical
   Assistance  :  Lawrence Bassham, Ron Boisvert, James Filliben,
Sharon Keller,
                  Daniel Lozier, and Bert Rust.

   Warning     :  Portability Issues.

   Limitation  :  Amount of memory allocated for workspace.

   Restrictions:  Permission to use, copy, and modify this software
without
```

```
                  fee is hereby granted, provided that this entire
notice is
                  included in all copies of any software which is or
includes
                  a copy or modification of this software and in all
copies
                  of the supporting documentation for such software.
------------------------------------------------------------------------
--- */

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include "../include/proto.h"

int main(int argc, char* argv[])
{
        FILE* fpGeneral, *fpGPBlockFreq, *fpGPLongestRuns, *fpInfile;
        int i, tb, blocklen;
        size_t bytesread;
        unsigned char* epsilon;
        struct stat* statInfo;

        /* Initialize. */
        fpGeneral = fpGPBlockFreq = fpGPLongestRuns = fpInfile = NULL;
        statInfo = NULL;
        epsilon = NULL;
        i = tb = blocklen = 0;
        bytesread = 0;

        fpGeneral = fopen("results", "w");
        if (fpGeneral == NULL)
        {
                perror("arjsts");
                printf("File for results could not be opened.
Exiting.\n");
                exit(1);
        }
        fpGPBlockFreq = fopen("gnuplot.bf", "w");
        if (fpGPBlockFreq == NULL)
        {
                perror("arjsts");
                printf("File for universal statistical test results
could not be opened. Exiting.\n");
                fclose(fpGeneral); fpGeneral = NULL;
                exit(1);
        }
        fpGPLongestRuns = fopen("gnuplot.lruns", "w");
        if (fpGPLongestRuns == NULL)
        {
                perror("arjsts");
                printf("File for longest runs test results could not be
opened. Exiting.\n");
                fclose(fpGeneral); fpGeneral = NULL;
```

```
                fclose(fpGPBlockFreq); fpGPBlockFreq = NULL;
                exit(1);
        }
        for (i=1; i<argc; i++)
        {
                fpInfile = fopen(argv[i], "r");
                if (fpInfile == NULL)
                {
                        perror("arjsts");
                        printf("File %s could not be opened.
Continuing.\n",
                                argv[i]);
                }
                else
                {
                        statInfo = (struct stat*)malloc(sizeof(struct
stat));
                        if (statInfo == NULL)
                        {
                                /* If we have no memory here, something
                                 * is definitely wrong. */
                                printf("Out of memory.\n");
                                exit(1);
                        }
                        if(stat(argv[i], statInfo))
                        {
                                /* Shouldn't ever happen -- we opened the
                                 * same file above. */
                                perror("arjsts");
                                printf("Error getting information for %s.
Continuing.\n", argv[i]);
                        }
                        else
                        {
                                /* Allocate the buffer for the bits
                                 * to be tested. */
                                epsilon = (unsigned char*)
                                        malloc(statInfo->st_size);
                                if (epsilon == NULL)
                                {
                                        printf("Not enough memory for
%s.\n",
                                                argv[i]);
                                        free(statInfo); statInfo = NULL;
                                        continue;
                                }

                                /* Record how many bits we use. */
                                tb = statInfo->st_size<<3;

                                /* Read bits to be tested into array. */
                                bytesread = fread(epsilon, 1,
                                        statInfo->st_size, fpInfile);
                                if (bytesread < statInfo->st_size)
                                {
                                        perror("arjsts");
```

```
                                        printf("Error reading %s.
Continuing.\n",
                                                argv[i]);
                                        free(statInfo); statInfo = NULL;
                                        free(epsilon); epsilon = NULL;
                                        continue;
                                }
                                /* Print headers. */
                                fprintf(fpGeneral, "------------------
\n");
                                fprintf(fpGeneral, "File: %s\n",
argv[i]);
                                fprintf(fpGeneral, "Size: %ld bytes\n",
                                        statInfo->st_size);

                                /* Decide how many bits in a block
                                 * for the block frequency test. */
                                if(statInfo->st_size <= 50000000)
                                        blocklen = 80000;
                                if(statInfo->st_size <= 5000000)
                                        blocklen = 8000;
                                if(statInfo->st_size <= 500000)
                                        blocklen = 800;
                                if(statInfo->st_size <= 50000)
                                        blocklen = 80;
                                if(statInfo->st_size > 50000000)
                                        blocklen = 800000;
                                BlockFrequency(fpGPBlockFreq, epsilon,
                                        blocklen, tb);
                                LongestRunOfOnes(fpGPLongestRuns,
epsilon, tb);
                                Runs(fpGeneral, epsilon, tb);
                                Universal(fpGeneral, epsilon, tb);
                                fprintf(fpGeneral, "\n");
                                fprintf(fpGPBlockFreq, "\n\n");
                                fprintf(fpGPLongestRuns, "\n\n");

                                free(epsilon); epsilon = NULL;
                        }

                        /* Clean up for the next loop. */
                        free(statInfo); statInfo = NULL;
                        fclose(fpInfile); fpInfile = NULL;
                }
        }

        /* Clean up for the end of the program. */
        fflush(NULL); /* Flush all open output streams. */
        fclose(fpGeneral); fpGeneral = NULL;
        fclose(fpGPBlockFreq); fpGPBlockFreq = NULL;
        fclose(fpGPLongestRuns); fpGPLongestRuns = NULL;

        return 0;
}
```

---

blockFrequency.c:

```
#include <stdio.h>
#include "../include/config.h"
#include "../include/proto.h"
#include "../include/mconf.h"

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * * * *
                    B L O C K  F R E Q U E N C Y  T E S T
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * * * */

void BlockFrequency(FILE* fpOut, unsigned char* epsilon, int m, int n)
{
  unsigned char testbyte;
  int    i, j, N, mByte;
  double blockSum, pi, v, sum;

  N = (int)floor((double)n/(double)m);/* # OF SUBSTRING BLOCKS */
  sum = 0.0;

  for(i = 0; i < N; i++) {     /* N=10000 FOR EACH SUBSTRING BLOCK */
     pi = 0.0;
     blockSum = 0.0;
     mByte = m >> 3; /* For speed in the loop. */
     for(j = 0; j < mByte; j++)       /* m=100 COMPUTE the "i"th Pi
Value */
     {
        /* Unroll the loop for performance, and because it's easier
         * to copy and paste than program another loop. Same deal for
         * the assignment to testbyte: faster and easier. */
        testbyte = epsilon[j+i*mByte];
        blockSum += testbyte & 0x01; testbyte >>= 1;
        blockSum += testbyte & 0x01; testbyte >>= 1;
        blockSum += testbyte & 0x01; testbyte >>= 1;
        blockSum += testbyte & 0x01; testbyte >>= 1;
        blockSum += testbyte & 0x01; testbyte >>= 1;
        blockSum += testbyte & 0x01; testbyte >>= 1;
        blockSum += testbyte & 0x01; testbyte >>= 1;
        blockSum += testbyte & 0x01;
     }
     pi = (double)blockSum/(double)m;
     v = pi - 0.5;
     fprintf(fpOut, "%d %f\n", i, v);
  }

  return;
}
```

---

longestRunOfOnes.c:

```
#include <stdlib.h>
#include <stdio.h>
#include "../include/proto.h"
#include "../include/mconf.h"
```

```
struct strecord
{
        unsigned int runlength;
        unsigned int tally;
        struct strecord* next;
};
typedef struct strecord record;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * *
                      L O N G E S T  R U N S  T E S T
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * */

void AddRun(record* chain, unsigned int length)
{
        record* temp;

        if(chain == NULL) return; /* These should never happen. */
        if(length == 0) return;

        while (chain->next != NULL && chain->next->runlength <= length)
                chain = chain->next;
        if(chain->runlength == length) chain->tally++;
        else
        {
                temp = (record*) malloc(sizeof(record));
                if(temp != NULL)
                {
                        temp->runlength = length; temp->tally = 1;
                        temp->next = chain->next;
                        chain->next = temp;
                }
                /* Otherwise, no memory, and ignore the result. */
        }
}

void LongestRunOfOnes(FILE* fpOut, unsigned char* epsilon, int n)
{
  record* zeros, *ones, *temp;
  unsigned char bits, last;
  unsigned int length, i, runsofzeros, runsofones;

  /* Sanity checking. */
  if(fpOut == NULL) return;
  if(epsilon == NULL) return;
  if(n < 1) return;

  /* Initialize. */
  zeros = (record*) malloc(sizeof(record));
  if(zeros == NULL) return; /* No memory. */
  ones = (record*) malloc(sizeof(record));
  if(ones == NULL)
  {
          /* No memory. */
          free(zeros);
          return;
```

```
}
zeros->runlength = 1; zeros->tally = 0; zeros->next = NULL;
ones->runlength = 1; ones->tally = 0; ones->next = NULL;
runsofzeros = runsofones = 0;
bits = epsilon[0]; length = 0; last = bits & 0x01;

/* The main loop where we count the runs and their lengths. The inner
 * loop has been unrolled for speed and ease of programming. */
for(i = 0; i < (n>>3); i++)
{
        bits = epsilon[i];

        if(last != (bits & 0x01))
        {
                last?AddRun(ones, length):AddRun(zeros, length);
                last?runsofones++:runsofzeros++;
                last = bits & 0x01; length = 1;
        }
        else length++;
        bits >>= 1;
        if(last != (bits & 0x01))
        {
                last?AddRun(ones, length):AddRun(zeros, length);
                last?runsofones++:runsofzeros++;
                last = bits & 0x01; length = 1;
        }
        else length++;
        bits >>= 1;
        if(last != (bits & 0x01))
        {
                last?AddRun(ones, length):AddRun(zeros, length);
                last?runsofones++:runsofzeros++;
                last = bits & 0x01; length = 1;
        }
        else length++;
        bits >>= 1;
        if(last != (bits & 0x01))
        {
                last?AddRun(ones, length):AddRun(zeros, length);
                last?runsofones++:runsofzeros++;
                last = bits & 0x01; length = 1;
        }
        else length++;
        bits >>= 1;
        if(last != (bits & 0x01))
        {
                last?AddRun(ones, length):AddRun(zeros, length);
                last?runsofones++:runsofzeros++;
                last = bits & 0x01; length = 1;
        }
        else length++;
        bits >>= 1;
        if(last != (bits & 0x01))
        {
                last?AddRun(ones, length):AddRun(zeros, length);
                last?runsofones++:runsofzeros++;
                last = bits & 0x01; length = 1;
```

```
          }
          else length++;
          bits >>= 1;
          if(last != (bits & 0x01))
          {
                  last?AddRun(ones, length):AddRun(zeros, length);
                  last?runsofones++:runsofzeros++;
                  last = bits & 0x01; length = 1;
          }
          else length++;
          bits >>= 1;
          if(last != (bits & 0x01))
          {
                  last?AddRun(ones, length):AddRun(zeros, length);
                  last?runsofones++:runsofzeros++;
                  last = bits & 0x01; length = 1;
          }
          else length++;
  }

  /* Last run. */
  last?AddRun(ones, length):AddRun(zeros, length);
  last?runsofones++:runsofzeros++;

  /* Write to the output file. */
  while(zeros != NULL)
  {
          fprintf(fpOut, "%d %f\n", zeros->runlength,
                          ((double)zeros-
>tally)/((double)runsofzeros)*100.0);
          temp = zeros; zeros = zeros->next;
          free(temp);
  }
  fprintf(fpOut, "\n\n");
  while(ones != NULL)
  {
          fprintf(fpOut, "%d %f\n", ones->runlength,
                          ((double)ones-
>tally)/((double)runsofones)*100.0);
          temp = ones; ones = ones->next;
          free(temp);
  }

  return;
}
```

---

runs.c:

```
#include <stdio.h>
#include <stdlib.h>
#include "../include/config.h"
#include "../include/proto.h"

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * *
                              R U N S   T E S T
```

```
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * */

void Runs(FILE* fpOut, unsigned char* epsilon, int n)
{
  int    i, nBytes;
  double V_n_obs;
  unsigned char testbyte;

  V_n_obs = 0;
  nBytes = n >> 3; /* Number of bytes. */
  for(i = 0; i < nBytes; i++)
  {
         /* XOR highlights the switches between 1 and 0. */
         testbyte = epsilon[i] ^ (epsilon[i]>>1);

         V_n_obs += testbyte & 0x01; testbyte >>= 1;
         V_n_obs += testbyte & 0x01; testbyte >>= 1;
         V_n_obs += testbyte & 0x01; testbyte >>= 1;
         V_n_obs += testbyte & 0x01; testbyte >>= 1;
         V_n_obs += testbyte & 0x01; testbyte >>= 1;
         V_n_obs += testbyte & 0x01; testbyte >>= 1;
         V_n_obs += testbyte & 0x01; testbyte >>= 1;

         if (i+1 < nBytes)
            V_n_obs += ((epsilon[i] >> 7) ^ epsilon[i+1]) & 0x01;
  }

  V_n_obs /= (double)n;
  fprintf(fpOut, "Normalized number of runs: %f\n", V_n_obs);

  return;
}
```

---

universal.c:

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "../include/config.h"
#include "../include/proto.h"

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * *
                         U N I V E R S A L  T E S T
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * */

void Universal(FILE* fpOut, unsigned char* epsilon, int n)
{
  int      i, j, p, K, L, Q, index;
  double   phi, sum;
  long*    T, decRep;
  unsigned char active, used;
```

```
  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * *
  * THE FOLLOWING REDEFINES L, SHOULD THE CONDITION:     n >=
1010*2^L*L       *
  * NOT BE MET, FOR THE BLOCK LENGTH L.
*
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * */

  /* Initialize. */
  L=0;

  /* Added by AJ. Minimum from NIST is L=6. */
  if (n >= 2020)       L = 1;
  if (n >= 8080)       L = 2;
  if (n >= 24240)      L = 3;
  if (n >= 64640)      L = 4;
  if (n >= 161600)     L = 5;
  /* End of AJ-added L values. */

  if (n >= 387840)     L = 6;
  if (n >= 904960)     L = 7;
  if (n >= 2068480)    L = 8;
  if (n >= 4654080)    L = 9;
  if (n >= 10342400)   L = 10;
  if (n >= 22753280)   L = 11;
  if (n >= 49643520)   L = 12;
  if (n >= 107560960)  L = 13;
  if (n >= 231669760)  L = 14;
  if (n >= 496435200)  L = 15;
  if (n >= 1059061760) L = 16;

  Q = 10*(int)pow(2,L);
  K = (int) (floor(n/L) - (double)Q); /* BLOCKS TO TEST */

  if ((L < 1) || (L > 16) || ((double)Q < 10*pow(2,L))) {
     printf("\t\tUNIVERSAL STATISTICAL TEST\n");
     printf("\t\t---------------------------------------\n");
     printf("\t\tERROR:  L IS OUT OF RANGE.\n");
     printf("\t\t-OR- :  Q IS LESS THAN %f.\n", 10*pow(2,L));
  }
  else {
     sum = 0.0;
     index = 0;
     used = 0;
     active = epsilon[index];
     p = (int)pow(2,L);
     T = (long*) calloc(p, sizeof(long));
     for(i = 0; i < p; i++) T[i] = 0;

     for(i = 1; i <= Q; i++) { /* INITIALIZE TABLE */
        decRep = 0;
        for(j = 0; j < L; j++)
        {
           decRep += (long)(active & 0x01) * (long)pow(2, j);
           active >>= 1; used++;
           if (used == 8) used=0,active=epsilon[++index];
```

```
        }
        T[decRep] = i;
     }

     for(i = Q+1; i <= Q+K; i++) { /* PROCESS BLOCKS */
        decRep = 0;
        for(j = 0; j < L; j++)
        {
           decRep += (long)(active & 0x01) * (long)pow(2, j);
           active >>= 1; used++;
           if (used == 8) used=0,active=epsilon[++index];
        }
        sum += log(i - T[decRep])/log(2);
        T[decRep] = i;
     }
     phi = (double)(sum/(double)K);

     fprintf(fpOut, "Result of universal statistical test: %f\n", phi);

     free(T);
  }
  return;
}
```

Makefile:

```
.SUFFIXES : .c .o
OBJS=arjassess.o blockFrequency.o longestRunOfOnes.c runs.o universal.o
CFLAGS=-Wall -O3 -c
EXEFLAGS=-Wall -O3
LDFLAGS=-lm -lc

.c.o :
       ${CC} ${CFLAGS} -o $@ $<

../arjassess : ${OBJS}
       ${CC} ${EXEFLAGS} -o $@ *.o ${LDFLAGS}
       /usr/bin/strip ../arjassess

clean:
       rm *.o ../arjassess
```

---

proto.h:

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * *
     S T A T I S T I C A L  T E S T  F U N C T I O N  P R O T O T Y P E
S
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * */

void BlockFrequency(FILE*, unsigned char*, int, int);
void LongestRunOfOnes(FILE*, unsigned char*, int);
void Runs(FILE*, unsigned char*, int);
void Universal(FILE*, unsigned char*, int);
```

Part II -- Network Detects

Detect number one

The following alert appeared in response to network traffic at the author's workplace one fine sunny day:

Jan 14 11:45:07 else.x.de snort: [ID 702911 auth.alert] [1:1384:2] MISC UPNP malformed advertisement [Classification: Misc Attack] [Priority: 2]: {UDP} 172.x.32.86:1900 -> 239.255.255.250:1900

This was immediately identified as a Windows XP machine, since Windows XP is the only operating system that the author is aware of that supports Universal Plug-n-Play, and in which it is enabled in the default installation. The UPnP Forum lists a boatload of members, but the UPnP Forum is Microsoft-created and Microsoft-sponsored (check the copyright notice on every page of the UPnP Forum). ("Universal ...") There were actually a series of alerts as this machine made its presence known to the network, and as it changed IP addresses. The first address was 192.168.0.1, which comes from a network that is completely separate from the company's main network. After that, it changed to the address listed above, which is an address assigned by the internal DHCP server, then disappeared from the network after a short time. The remaining alerts are not listed above, because they are identical to the first except for the IP address.

After investigation, the offending laptop was found, already disconnected from the network, and was thoroughly scanned. There was nothing wrong with the laptop, and the packets observed earlier could not be reproduced under test conditions. The suspicion is that the user unwittingly did something that caused a UPnP advertisement to be sent, and the Snort alarm was a false alarm. There was a lot of unnecessary software installed on the laptop, which supports the assertion that it could at some point start advertising something it had not mentioned before, probably as the result of some action on the part of the user. The laptop was from Dell, and it seems like absolutely everything was enabled. Spanning tree, of all things, was running on it.

1. **Source of trace:** This trace was found in the internal network of the author's employer. The network structure is flat, and most addresses come from the private class B network 172.x.0.0/16. There are some other private network numbers attached to this network, but they are separated from the main network with routers.
2. **Detect was generated by:** Snort 1.9.0 with the latest rulesets as of the date of the attack. The date and time are accurate in the trace.
3. **Probability the source address was spoofed:** Vanishingly small. The laptop that originated these packets was found, and the MAC address of the Ethernet card was compared to that of the packets found during the detect. While it is possible to spoof a MAC address as well, there are only a handful of computers on our network running Windows XP, and most of those were installed with our company's automatic installation method, a part of which is to turn off SSDP,

upon which UPnP relies. The packets also appeared during the time that the laptop was connected to our network, and not at other times. Finally, during the incident, we were able to trace the origin of the packets back to the building where the laptop was plugged in by checking the switches along the path to determine where the MAC address was coming from. We were not able to trace it back to the exact switch port before the laptop was pulled from the network.

4. **Description of attack:** The Snort rule that generated this attack is:

   alert udp $EXTERNAL_NET any -> $HOME_NET 1900 (msg:"MISC UPNP malformed advertisement"; content:"NOTIFY * "; nocase; classtype:misc-attack; reference:cve,CAN-2001-0876; reference:cve,CAN-2001-0877; sid:1384; rev:2;).

   The two referenced vulnerabilities in the CVE list are a buffer overflow in the "Location" parameter of a UPnP advertisement and a potential denial of service, also related to the "Location" parameter. It must however be noted that the Snort rule that generated this alert is not precise, and will fire on every UPnP advertisement. The author sent an e-mail concerning this and a possible misordering of two of the UPnP rules in the standard Snort signatures to the Snort signatures e-mail list, but never received a reply.

5. **Attack mechanism:** The "Location" parameter of a UPnP announcement is trusted implicitly in Microsoft's implementation of UPnP, leading to the possibility of specifying a malicious URL. From the eEye advisory:

   A malicious attacker could specify a chargen service on a remote machine causing the XP client to connect and get caught in a tight read/malloc loop. Doing this will throw the machine into an unstable state where CPU utilization is at %100 [sic] and memory is being allocated to the point that it is totally consumed. This basically makes the remote XP system completely unusable and requires a physical power-off shutdown. (Hassell)

   The other vulnerability is a buffer overflow which leads to code execution in the SYSTEM context. Both are very serious vulnerabilities, because they can be executed against a complete network by sending a single multicast packet.

6. **Correlations:** No reports of this attack in the wild are forthcoming, only thousands of security advisories and news articles. There is also a GCIH practical available at http://www.giac.org/practical/Chip_Calhoun_GCIH.doc that gives more details on the vulnerabilities, available exploits, and defensive measures. (Calhoun) This attack is very well known, as it was the first big vulnerability discovered in Windows XP. The CVE assignments CAN-2001-0876 and CAN-2001-0877 were mentioned above.
7. **Evidence of active targeting:** This is not active targeting. It targets an entire network at one time, because it's sent to a multicast address.
8. **Severity:** severity = (criticality + lethality) - (system countermeasures + network countermeasures).

Criticality: This rates as a 3. In our network, we only have a handful of Windows XP computers, and they are exclusively desktop systems for users. We have no Windows ME machines, and no networked Windows 98 machines to the best of the author's knowledge.

Lethality: This rates as a 5. The buffer overflow attack would immediately compromise all listening machines, and the denial of service attack would require all of them to be rebooted.

System countermeasures: This is a 4. The automated installation used at the author's place of employment turns SSDP off. There are a couple of computers on the network that were installed by hand. In general, the administrators of the client machines are behind on installing patches, but they seem to be doing a better job with the few Windows XP installations we have.

Network countermeasures: This is a 1. Almost the entire network is flat, which only helps this kind of attack.

All in all, we have (3+5) - (4+1) = 3, which warrants attention.

9. **Defensive recommendation:** The one big network should be broken out into a few smaller networks by department or some other meaningful criterion. All computers installed manually with Windows XP should be found, SSDP and UPnP should be turned off, and all future installations of client machines should happen centrally using a well-hardened installation. An important factor of the standard installation should be turning off unneeded services.
10. **Multiple choice test question:** The two known vulnerabilities in the Microsoft implementation of UPnP/SSDP referenced in CAN-2001-0876 and CAN-2001-0877 use what mechanism?
    a. The UPnP reserved flags bits
    b. The "Location" tag
    c. The broadcast address
    d. The vulnerabilities are in the core protocol

    The correct answer is "b".

Detect number two

This is a detect from in front of the corporate firewall at the author's place of employment. About four hours worth of data, or 2 gigabytes, were collected. IP addresses for the author's company have been changed, but the "attacking" IP address is the original one. The 2 gigabytes of data were plugged through Snort 1.9.0 with the most recent ruleset as of the 17th of January 2003. The Snort rulebase and configuration file was somewhat diminished in order to reduce the huge number of false positives and uninteresting traffic. The detect is not an alert, but rather an alleged port scan:

01/17-14:20:59.534419 TCP src: x.x.x.9 dst: 195.19.9.56 sport: 3054 dport: 41333 tgts: 7 ports: 27 flags: ******S* event_id: 76
01/17-14:22:00.910313 ICMP src: x.x.x.9 dst: 195.19.9.56 type: 3 code: 3 tgts: 7 event_id: 76

This raises eyebrows for a couple of reasons. The destination port of 41333 looks contrived. In fact, this could be interpreted as a cartoonish "AIEEE!!!", which was the administrator's initial reaction upon investigating this detect, until he discovered that it was mostly harmless. The ICMP response about a port being unreachable (type 3 code 3) a minute later is also out of the ordinary.

1. **Source of trace:** These data were collected between the router belonging to the author's company's ISP and the company's firewall. There is also an HTTP proxy in that transit network with the IP address x.x.x.9. There isn't much besides that in the network.
2. **Detect was generated by:** Snort 1.9.0 with the most recent ruleset as of the 17th of January 2003. The command line used to run Snort was:
   snort -d -l gcia_log -c /usr/local/etc/snort.conf -h x.x.x.0/24 -r gcia.cap
   The port scan plugin generated the detect.
3. **Probability the source address was spoofed:** None. The source address is our proxy, the trace comes from the same switch as the proxy, there is a complete TCP session between the two computers in question, and the following entry appears in the proxy's log:

```
4.
5.  1042802282.451  73699 172.x.6.9 TCP_MISS/503 1187 GET
   ftp://195.19.9.56:41333/tsrh/reliz/2002/oct/tsrh-
   extremepicturefinder143_oct_06.zip - DIRECT/195.19.9.56
```

   172.x.6.9 is the address of the internal proxy (behind the firewall).

6. **Description of attack:** At this point we need a more complete view of the attack to understand why this has been singled out for analysis. The command line used to extract the trace was:
   tcpdump -n -vv -X -r gcia.cap host 195.19.9.56
   The slightly abridged session:

```
7.
8.  14:20:59.534419 x.x.x.9.3054 > 195.19.9.56.41333: S [tcp sum ok]
   1980890228:1980890228(0) win 32120 <mss 1460,sackOK,timestamp
   1271790402 0,nop,wscale 0> (DF) (ttl 64, id 60439, len 60)
9.  [hex deleted]
10. 14:21:00.379079 195.19.9.56.41333 > x.x.x.9.3054: S [tcp sum ok]
   113270201:113270201(0) ack 1980890229 win 8760 <mss 1460> (DF)
   (ttl 112, id 57574, len 44)
11. [hex deleted]
12. 14:21:00.379210 x.x.x.9.3054 > 195.19.9.56.41333: . [tcp sum ok]
   1:1(0) ack 1 win 32120 (DF) (ttl 64, id 61007, len 40)
13. [hex deleted]
14. 14:22:00.893896 195.19.9.56.41333 > x.x.x.9.3054: P [tcp sum ok]
   1:81(80) ack 1 win 8760 (DF) (ttl 112, id 15085, len 120)
```

```
15.  0x0000   4500 0078 3aed 4000 7006 xxxx c313 0938
     E..x:.@.p.xx...8
16.  0x0010   xxxx xx09 a175 0bee 06c0 5dba 7611 fc75
     xxx..u....].v..u
17.  0x0020   5018 2238 1d82 0000 3232 3020 2e2e 2e20
     P."8....220.....
18.  0x0030   4672 6f6d 2064 7265 616d 2074 6f20 6472
     From.dream.to.dr
19.  0x0040   6561 6d20 7765 2068 6176 6520 616c 7761
     eam.we.have.alwa
20.  0x0050   7973 2062 6565 6e20 6c69 6b65 2061 6e20
     ys.been.like.an.
21.  0x0060   6576 6572 2066 6c6f 7769 6e67 2073 7472
     ever.flowing.str
22.  0x0070   6561 6d2e 2e2e 0d0a                     eam.....
23.  14:22:00.893974 x.x.x.9.3054 > 195.19.9.56.41333: . [tcp sum ok]
   1:1(0) ack 81 win 32120 (DF) (ttl 64, id 10926, len 40)
24.  [hex deleted]
25.  14:22:00.894520 x.x.x.9.3054 > 195.19.9.56.41333: P [tcp sum ok]
   1:17(16) ack 81 win 32120 (DF) (ttl 64, id 10928, len 56)
26.  0x0000   4500 0038 2ab0 4000 4006 xxxx xxxx xx09
     E..8*.@.@.xxxxx.
27.  0x0010   c313 0938 0bee a175 7611 fc75 06c0 5e0a
     ...8...uv..u..^.
28.  0x0020   5018 7d78 54b0 0000 5553 4552 2061 6e6f
     P.}xT...USER.ano
29.  0x0030   6e79 6d6f 7573 0d0a                     nymous..
30.  14:22:00.910229 195.19.9.56.137 > x.x.x.9.137: [udp sum ok]
31.  >>> NBT UDP PACKET(137): QUERY; REQUEST; BROADCAST
32.  TrnID=0xC1E0
33.  OpCode=0
34.  NmFlags=0x1
35.  Rcode=0
36.  QueryCount=1
37.  AnswerCount=0
38.  AuthorityCount=0
39.  AddressRecCount=0
40.  QuestionRecords:
41.  Name=*               NameType=0x00 (Workstation)
42.  QuestionType=0x21
43.  QuestionClass=0x1
44.
45.   (ttl 112, id 16109, len 78)
46.  0x0000   4500 004e 3eed 0000 7011 xxxx c313 0938
     E..N>...p.xx...8
47.  0x0010   xxxx xx09 0089 0089 003a 2d3c c1e0 0010
     xxx......:-<....
48.  0x0020   0001 0000 0000 0000 2043 4b41 4141 4141
     .........CKAAAAA
49.  0x0030   4141 4141 4141 4141 4141 4141 4141 4141
     AAAAAAAAAAAAAAAA
50.  0x0040   4141 4141 4141 4141 4100 0021 0001
     AAAAAAAAA..!..
51.  14:22:00.910313 x.x.x.9 > 195.19.9.56: icmp: x.x.x.9 udp port
   137 unreachable for 195.19.9.56.137 > x.x.x.9.137: [udp sum ok]
52.  [A few more occurrences of the same two packets deleted]
```

```
53.  14:22:02.779747 195.19.9.56.41333 > x.x.x.9.3054: F [tcp sum ok]
   122:122(0) ack 17 win 8744 (DF) (ttl 112, id 32237, len 40)
54.  [hex deleted]
55.  14:22:02.779813 x.x.x.9.3054 > 195.19.9.56.41333: . [tcp sum ok]
   17:17(0) ack 81 win 32120 (DF) (ttl 64, id 11330, len 40)
56.  [hex deleted]
57.  [Another few occurrences of the NetBIOS and ICMP packets
   deleted]
58.  14:22:04.930748 195.19.9.56.41333 > x.x.x.9.3054: FP [tcp sum
   ok] 81:122(41) ack 17 win 8744 (DF) (ttl 112, id 51181, len 81)
59.  0x0000   4500 0051 c7ed 4000 7006 xxxx c313 0938
     E..Q..@.p.xx...8
60.  0x0010   xxxx xx09 a175 0bee 06c0 5e0a 7611 fc85
     xxx..u....^.v...
61.  0x0020   5019 2228 1465 0000 3432 3120 3136 206f
     P."(.e..421.16.o
62.  0x0030   7574 2031 3620 7573 6572 7320 7265 6163
     ut.16.users.reac
63.  0x0040   6865 642e 2054 7279 206c 6174 6572 2e0d
     hed..Try.later..
64.  0x0050   0a                                              .
65.  14:22:04.930807 x.x.x.9.3054 > 195.19.9.56.41333: . [tcp sum ok]
   17:17(0) ack 123 win 32120 (DF) (ttl 64, id 11470, len 40)
66.  [hex deleted]
67.  14:22:04.931860 194.173.66.9.3054 > 195.19.9.56.41333: F [tcp
   sum ok] 17:17(0) ack 123 win 32120 (DF) (ttl 64, id 11471, len
   40)
68.  [hex deleted]
69.  14:22:05.861098 195.19.9.56.41333 > 194.173.66.9.3054: . [tcp
   sum ok] 123:123(0) ack 18 win 8744 (DF) (ttl 112, id 59629, len
   40)
70.  [hex deleted]
```

The first three packets aren't interesting. They're just a standard TCP threeway handshake.

In the fourth packet we see the quote "From dream to dream we have always been like an ever flowing stream". What on earth is that? Google tells us that this comes from the lyrics to a song by the heavy metal band Dismember. Suspicion is called for if the administrators of an FTP server use a non-standard port and have lyrics from a heavy metal song as the banner for their FTP site. No offense to the heavy metal fans out there.

Incidentally, how do we know that this is FTP? First and foremost, it was listed as such in the proxy log. Second, we see the 220 status code in the fourth packet and other FTP status codes in following packets.

Now that the banner has been sent, the proxy replies immediately with an anonymous logon attempt.

The FTP server initiates a scan of the proxy for the NetBIOS name service. Why would it do that? Passive fingerprinting tells us that this is likely to be a Windows

2000 Professional machine (win 8760, ttl about 128, "don't fragment" bit set, and only the maximum segment size option set), which means it's entirely possible that it's simply trying to get information from the proxy that it feels it needs or wants. In fact, this is probably name resolution. If a Windows machine is presented with an IP address for which it has no name, it sends exactly such a query to the unknown machine. Note also, that the server seems to ignore the ICMP responses. Perhaps the messages get lost on the way, perhaps there is a firewall on the server side, or perhaps this is simply poor programming on the part of Microsoft. Without a Windows 2000 Professional machine to test, it is hard to know. Another possible explanation for the NetBIOS scans, if we swing into paranoia mode, is that the server is looking for information, and is possibly looking to hide behind an existing connection to attempt to evade filtering in order to get it. Through the NetBIOS name service, it is possible to read the NetBIOS table of the target host, including the NetBIOS name of the destination computer, the names of logged in users, and the name of the workgroup or domain. This is an especially valuable information source if the target machine is the master browser for the workgroup or domain. The NetBIOS request detected here is a wildcard request, which would read all of those things out of the NetBIOS table of the target computer. One last possibility begs to be mentioned. Many IRC servers scan a connecting computer to determine if it is an open proxy. This may be a scan of some kind to check if sufficient security measures are in place at the originating site. Exactly what use this information could be to the FTP administrator is unclear. This possibility is not likely, and is only mentioned for the sake of completeness.

It is interesting to note that neither the NetBIOS scans nor the window size of 8760 could be reproduced from the author's home network when he tried connecting to the same FTP server the next morning. The server now uses a window size of 8472 and waits one minute between the TCP three-way handshake and the FTP logon banner, implying that it may still be waiting on a response to the NetBIOS packets, but the administrator may have installed a firewall (and perhaps a quality of service device that works by lowering the TCP window size) between the first detect and later attempts. The detect was on a Friday afternoon and the following attempts were the next day. Friday afternoon or evening would be a normal maintenance window. This would perhaps indicate that the FTP server is in some way legitimate if a firewall was installed but port 41333 is still open. If not that, then the firewall administrator took a default open stance and only blocked things like the NetBIOS ports.

Finally, back in our trace, the FTP server responds, telling our client that the maximum anonymous user limit has been reached, a whopping 16 users. What kind of FTP site limits their anonymous logons to 16 users? A very small one.

Besides the NetBIOS scans, what makes this detect worth noting and worrying about? Suspicion has already been cast on the quality of this FTP site, and a Google search for the IP address reveals a number of pages, mostly in Russian,

that claim software can be downloaded from this FTP server on port 41333. The software that one of our users tried to access is trialware. The full version can be purchased for about $30. ("Extreme Picture Finder...") The software can be downloaded from the Web page belonging to the developer of the software, Extreme Internet Software. The IP address we discovered is not listed as a mirror site, and at a limit of 16 anonymous users, it could hardly qualify to be one. According to RIPE, the IP address belongs to the Moscow State University of Geodesy and Cartography. Would a university represent itself on a public FTP server with nothing but a quote from a heavy metal song, and on a non-standard port, no less? The concern here is obvious: this could be a warez site, and/or an FTP site with trojanized copies of software. Either would be a problem for the author's company; one would be a policy problem, and the other would most definitely be a technical security problem.

71. **Attack mechanism:** Scanning for the NetBIOS name service is a common tactic, and can give an attacker information, such as the names of users on the target machine, which can then be used to brute force passwords. A NetBIOS scan also identifies poorly protected Internet hosts that could be ripe for cracking. Trojanized software is also a well-known attack vector and needs no further commentary.
72. **Correlations:** A Google search for the IP address and the results of the search were mentioned in the last section. The same search also turned up logs from a proxy that included downloads from the suspect FTP server. ("NDHU Proxy ...", Note: It was probably a mistake that this Web page was available while this research was being conducted. Access control lists have since been put in place, making it impossible to view this reference.) Scans of the NetBIOS name service port have long been in the top scanned ports listed on DShield's Web site. ("DShield Port Report...")
73. **Evidence of active targeting:** This is more like reactive targeting. Our proxy made the first connection, and the FTP server scanned us in response. There was no active targeting here.
74. **Severity:** severity = (criticality + lethality) - (system countermeasures + network countermeasures).

    Criticality: 3. This involves one end system and the proxy to the Internet. The loss of the end system would not be a big deal. The proxy would be more of a problem, but most business processes would not truly be adversely affected. A backup plan exists should the proxy fail, and the Web would be available again in a matter of 10 minutes to half an hour.

    Lethality: 2. The NetBIOS scan is nothing more than information gathering. If the software is trojanized, that could be somewhat more lethal, but this attempt to download software also failed.

System countermeasures: 2. The proxy is mostly unpatched and unhardened. The end system has virus filtering, but it would not necessarily recognize a trojan horse. Nothing like a sandbox is installed on the end system.

Network countermeasures: 2. The proxy is in front of the firewall, unprotected. There is no content filtering between server and end system that would try to detect and remove trojanized code, or even any executables, from the data stream. The end system is well hidden behind a firewall.

This leaves us with a severity of (3+2)-(2+2)=1, meaning the author did not run back in to work on Saturday to verify the integrity of all affected systems, but rather that he's planing on taking some countermeasures in the coming months.

75. **Defensive recommendation:** The proxy definitely needs to be in a zone protected by the firewall. The proxy administrator should evaluate the feasibility of allowing FTP requests only to the standard FTP command port. Content filtering somewhere in the proxy chain would also be a very big plus. Finally, if end systems could be installed with some kind of sandbox technology or be so configured that they only start programs that the administrator allows, attacks from possible trojans could be discounted in the future.
76. **Multiple choice test question:** What makes the following trace noteworthy?

```
77.
78.  14:22:04.930748 195.19.9.56.41333 > x.x.x.9.3054: FP [tcp sum
   ok] 81:122(41) ack 17 win 8744 (DF) (ttl 112, id 51181, len 81)
79.  0x0000   4500 0051 c7ed 4000 7006 71b7 c313 0938
     E..Q..@.p.q....8
80.  0x0010   xxxx xx09 a175 0bee 06c0 5e0a 7611 fc85
     xxx..u....^.v...
81.  0x0020   5019 2228 1465 0000 3432 3120 3136 206f
     P."(.e..421.16.o
82.  0x0030   7574 2031 3620 7573 6572 7320 7265 6163
     ut.16.users.reac
83.  0x0040   6865 642e 2054 7279 206c 6174 6572 2e0d
     hed..Try.later..
84.  0x0050   0a                                              .
```

   a. One would seldom expect to see an IP ID of 51181 in the wild.
   b. The FIN and PUSH flags are both set.
   c. The IP version number is not normal.
   d. This appears to be an FTP command channel on a non-standard port.

   The correct answer is "d".

Detect number three

Leah, this analysis is dedicated to You. Your daddy loves You. :)

This detect was submitted to intrusions@incidents.org twice. No responses were received. The original post can be found at http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00290.html. The repost, which was merely the

previous link plus a repeated request for commentary and questions, can be found at http://cert.uni-stuttgart.de/archive/intrusions/2003/02/msg00007.html. Because no responses were received, this detect is nearly identical to the first post, except for a couple of very small additions and improvements that the author thought of while proofreading, and except for the HTML formatting and a few spelling corrections.

Detects from Snort:

```
[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
05/14-20:45:01.104488 194.78.59.253:80 -> 78.37.135.141:80
TCP TTL:39 TOS:0x0 ID:52989 IpLen:20 DgmLen:40
***A**** Seq: 0xF2  Ack: 0x0  Win: 0x578  TcpLen: 20
--
[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
05/14-20:45:05.974488 194.78.59.253:80 -> 78.37.135.141:80
TCP TTL:39 TOS:0x0 ID:53239 IpLen:20 DgmLen:40
***A**** Seq: 0x14F  Ack: 0x0  Win: 0x578  TcpLen: 20
--
[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
05/14-20:45:10.964488 212.88.236.2:80 -> 78.37.135.141:80
TCP TTL:39 TOS:0x0 ID:53491 IpLen:20 DgmLen:40
***A**** Seq: 0x1AE  Ack: 0x0  Win: 0x578  TcpLen: 20
--
[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
05/14-20:45:15.924488 212.88.236.2:80 -> 78.37.135.141:80
TCP TTL:39 TOS:0x0 ID:53747 IpLen:20 DgmLen:40
***A**** Seq: 0x20E  Ack: 0x0  Win: 0x578  TcpLen: 20
--
[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
05/14-20:48:54.654488 61.218.166.98:80 -> 78.37.136.232:80
TCP TTL:44 TOS:0x0 ID:9054 IpLen:20 DgmLen:40
***A**** Seq: 0x3EF  Ack: 0x0  Win: 0x578  TcpLen: 20
--
[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
05/14-20:48:59.664488 61.218.166.98:80 -> 78.37.136.232:80
TCP TTL:44 TOS:0x0 ID:9650 IpLen:20 DgmLen:40
***A**** Seq: 0x5E  Ack: 0x0  Win: 0x578  TcpLen: 20
--
[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
05/14-20:49:04.774488 61.218.166.106:80 -> 78.37.136.232:80
TCP TTL:44 TOS:0x0 ID:10234 IpLen:20 DgmLen:40
***A**** Seq: 0xCE  Ack: 0x0  Win: 0x578  TcpLen: 20
--
[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
05/14-21:38:52.724488 64.152.70.68:80 -> 78.37.212.28:53
TCP TTL:49 TOS:0x0 ID:55639 IpLen:20 DgmLen:40
```

```
***A**** Seq: 0x38C  Ack: 0x0  Win: 0x578  TcpLen: 20
--
[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
05/14-21:38:52.724488 64.152.70.68:53 -> 78.37.212.28:53
TCP TTL:49 TOS:0x0 ID:55640 IpLen:20 DgmLen:40
***A**** Seq: 0x38D  Ack: 0x0  Win: 0x578  TcpLen: 20
--
[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
05/14-21:38:52.814488 63.211.17.228:80 -> 78.37.212.28:53
TCP TTL:49 TOS:0x0 ID:13980 IpLen:20 DgmLen:40
***A**** Seq: 0x3C9  Ack: 0x0  Win: 0x578  TcpLen: 20
```

Corresponding packets from tcpdump:

```
tcpdump -r 2002.4.14 -S -v 'tcp[8:4] = 0 and tcp[13] = 0x10'
20:45:01.104488 194.78.59.253.http > 78.37.135.141.http: . [bad tcp
cksum b8b8!] ack 0 win 1400 (ttl 39, id 52989, len 40, bad cksum 381c!)
20:45:05.974488 194.78.59.253.http > 78.37.135.141.http: . [bad tcp
cksum b8b8!] ack 0 win 1400 (ttl 39, id 53239, len 40, bad cksum 3722!)
20:45:10.964488 unknown-212.88.236.2.codenet.be.http >
78.37.135.141.http: . [bad tcp cksum b8b8!] ack 0 win 1400 (ttl 39, id
53491, len 40, bad cksum 7416!)
20:45:15.924488 unknown-212.88.236.2.codenet.be.http >
78.37.135.141.http: . [bad tcp cksum b8b8!] ack 0 win 1400 (ttl 39, id
53747, len 40, bad cksum 7316!)
20:48:54.654488 61-218-166-98.HINET-IP.hinet.net.http >
78.37.136.232.http: . [bad tcp cksum bab7!] ack 0 win 1400 (ttl 44, id
9054, len 40, bad cksum f86d!)
20:48:59.664488 61-218-166-98.HINET-IP.hinet.net.http >
78.37.136.232.http: . [bad tcp cksum bab7!] ack 0 win 1400 (ttl 44, id
9650, len 40, bad cksum f619!)
20:49:04.774488 61-218-166-106.HINET-IP.hinet.net.http >
78.37.136.232.http: . [bad tcp cksum bab7!] ack 0 win 1400 (ttl 44, id
10234, len 40, bad cksum f3c9!)
21:38:52.724488 proximitycheck2.allmusic.com.http >
78.37.212.28.domain: . [bad tcp cksum b7ba!] ack 0 win 1400 (ttl 49, id
55639, len 40, bad cksum 4ca3!)
21:38:52.724488 proximitycheck2.allmusic.com.domain >
78.37.212.28.domain: . [bad tcp cksum b7ba!] ack 0 win 1400 (ttl 49, id
55640, len 40, bad cksum 4ca2!)
21:38:52.814488 proximitycheck1.allmusic.com.http >
78.37.212.28.domain: . [bad tcp cksum b7ba!] ack 0 win 1400 (ttl 49, id
13980, len 40, bad cksum 2484!)
```

1. **Source of detect:** http://www.incidents.org/logs/Raw/2002.4.14 sanitized as described in http://www.incidents.org/logs/Raw/README. This file was chosen because it comes from the date closest to the expected date of arrival of my first child, Leah.

   An excellent analysis of the network structure of the target network was done by André Cormier in his GCIA practical post. (Cormier) The two MAC addresses

identified by André were verified as being the only two present in this log as well, so the physical structure should look the same as outlined by André.

Using a slight modification to André's techniques for identifying IP addresses talking to each other in this log file, we look for destination IP addresses on both sides of the IDS system. We want destination IP addresses because this gives us a better indication of the routing than looking for source addresses. Source addresses can be spoofed, we would expect to see the most attacks coming from the outside directed at inside addresses, including inactive addresses, and we want to identify what addresses are to be considered inside addresses.

```
tcpdump -enr 2002.4.14 'ether dst 0:0:c:4:b2:33' | cut -f8 -d" "
| cut -f1-4 -d. | sort -u
[345 addresses from the 78.37.0.0/16 network]
tcpdump -enr 2002.4.14 'ether dst 0:3:e3:d9:26:c0' | cut -f8 -d"
" | cut -f1-4 -d. | sort -u
[24 addresses from many different networks except 78.37.0.0/16]
```

Thus, the inside network is 78.37.0.0/16, and this appears to be that network's Internet connection (or one of them).

2. **Detect was generated by:** The original detect was found and logged by an unknown version of Snort with an unknown ruleset. This detect was generated by Snort 1.9.0 with the latest ruleset as of the 24th of January 2003. The rule that caused the alarm to fire is:

```
3.
4.  alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN nmap
TCP";flags:A;ack:0; reference:arachnids,28; classtype:attempted-
recon; sid:628; rev:1;)
```

This detects TCP packets that have the ACK flag and only the ACK flag set with an acknowledgement number of zero. An acknowledgement number of zero is very rare. It was pointed out in a post from Ashley Thomas to intrusions@incidents.org that Linux normally uses an initial sequence number of zero (Thomas), and thus one would expect to see an acknowledgement number of zero during the second step of the TCP three-way handshake. In that case, the SYN flag would also be set. The only way the combination detected by this Snort rule can occur naturally is when the TCP sequence numbers wrap and happen to land on zero.

5. **Probability the source address was spoofed:** Low for some, high for others. This is a mapping attempt, and not receiving a reply because the source address is spoofed would nullify the reason for sending the packet in the first place. The packets from Allmusic are for the purposes of improving download speeds, and they are almost certainly not spoofed. However, looking at the first four packets we see that the same host is scanned from two different sources using the same method, and both within mere seconds of each other. In fact, they are sent in

approximately 5 second intervals. This is probably someone who wants to hide his mapping attempt by including a decoy address. The probability of one of those two addresses being spoofed is high. One might be tempted to say that the third scan belongs with the first two, simply because it's only three minutes later. This is nonetheless unlikely. Looking at the patterns we can see that a different host is being scanned, and that there are three retries as opposed to two retries each from the previous two scans. There is also no decoy here, although the target host is different, so the attacker would get no response if the source address were spoofed.

6. **Description of attack:** Most of these are mapping attempts, probably using the tool nmap ("Nmap - ..."). The last three packets are an attempt to find the server closest to the client for improved download performance. The nmap command line used for the real mapping attempts would be:
7.
8. `nmap -sA -g 80 -p 80 [-D decoy_IP] target_IP`
9. **Attack mechanism:** The idea behind the first seven packets is to send a packet claiming to be part of an active TCP session to an address in the hopes of eliciting a RST. A response of this kind indicates that there is a machine belonging to the target address. A lack of response means either that the packets are being filtered or that the address does not belong to a computer.

   The last three packets do not appear to be mapping attempts, but rather, based on the DNS names given the machines, seem to be computers with the task of determining how close the computer requesting a download is to various Allmusic servers. The purpose would be to attempt to find the closest server to improve download times. The names proximitycheck1.allmusic.com and proximitycheck2.allmusic.com do not resolve to addresses, but looking up their addresses from the detects in the ARIN database tells us that the address for proximitycheck2 is in the same address block as the Web server www.allmusic.com. The owner of that address block (Level 3 Communications, Inc.) also owns the address block with proximitycheck1, and the nearly simultaneous arrival times of the last three packets lead us to believe that both servers are legitimate, and are simply doing a proximity check. This, then, is a false alarm.

   These packets all attempt to get through packet filtering in the same way. The real mapping attempts all come from port 80 and go to port 80, since port 80 (for HTTP) is one of the most likely ports to be open. If port 80 as a destination port is not open at the receiving end, it may well be that a simple packet filter that lacks any kind of stateful inspection capabilities will view this as a response to a previous HTTP request from an internal client because it is an acknowledgement, and it comes from the source port 80. Allmusic uses a similar technique, but they use the destination port 53 (for DNS) and the source ports 53 and 80, again figuring that one of these combinations may be allowed through any packet filtering devices that may be in place.

10. **Correlations:** The mapping attempts are well known, and are described in the manual page for nmap. (Fyodor) There does not appear to be any information related to the proximity checks on www.allmusic.com. Such load balancing techniques are well known and are used by other companies for similar purposes. There is also one occurrence of the same alarm from proximitycheck2.allmusic.com in a Snort log for the 27th of September 2002 for someone at the University of Denver who posts her/his logs on the Web. ("Snort Report")
11. **Evidence of active targeting:** Only two hosts are scanned from three addresses, and we have seen that one of those scanning attempts was most likely a decoy. This is active targeting.
12. **Severity:** severity = (criticality + lethality) - (system countermeasures - network countermeasures).

    Criticality: 3. We do not have complete log files, but considering the lack of any other traffic to or from the targeted hosts, they are not all that likely to be critical hosts. We might expect to see more attacks against a critical host. It is also unlikely that someone would be downloading music from a critical host unless this were a proxy or network address translation device with many-to-one NAT. On the other hand, active targeting of a non-critical host doesn't make a whole lot of sense. So, we pick a number in the middle.

    Lethality: 1. This is nothing more than information gathering, and it doesn't yield much information at that.

    System countermeasures: 2. This is completely unknown, but we err on the side of caution without assuming the worst.

    Network countermeasures: 2. Same reasoning as for system countermeasures. The only other thing we can add is that there are attacks from the source address 255.255.255.255 in the log. We don't know if those attacks got past the internal router, but the external router has no spoofing protection or sanity checking.

    This leaves us with (3+1)-(2+2)=0. Most administrators would ignore this attack.

13. **Defensive recommendation:** If a stateful firewall is not already in use, it should be. A proper stateful firewall is all that is really needed to block this attack. If that is not possible, at least being precise in writing the filter rules is recommended. For example, it is safe to specify that if the source (destination) port is 80, then the destination (source) port has to be greater than 1023. Additionally, if the external router is under the administration of the site's administrators (and not those of the ISP, for instance), doing basic spoofing protection on incoming (and outgoing) packets would be good. As already mentioned, packets coming from the source address 255.255.255.255 should never make it through border routers into the destination site. This spoofing protection should include blocking private IP address spaces, broadcast addresses, loopback addresses, and multicast addresses

if they are not in use at the site. The external interface of the border router should make certain that no packets arrive destined for the internal network with a source address belonging to the internal network. Similarly, the internal interface of the border router should make certain that no packets leave the internal network with a source address other than one from the internal network, or rather from an official, routable IP address assigned to the organization in question. Internal private addresses should also be blocked from leaving the network.

14. **Multiple choice question:** What is most likely true about the following detects:

```
15.
16.  20:45:01.104488 194.78.59.253.http > 78.37.135.141.http: . ack 0
   win 1400 (ttl 39, id 52989, len 40)
17.  20:45:05.974488 194.78.59.253.http > 78.37.135.141.http: . ack 0
   win 1400 (ttl 39, id 53239, len 40)
18.  20:45:10.964488 212.88.236.2.http > 78.37.135.141.http: . ack 0
   win 1400 (ttl 39, id 53491, len 40)
19.  20:45:15.924488 212.88.236.2.http > 78.37.135.141.http: . ack 0
   win 1400 (ttl 39, id 53747, len 40)
```

   a. They would elicit a SYN/ACK from the receiving host.
   b. One of the source addresses is probably spoofed.
   c. They probably come from a Linux computer because the acknowledgement number is zero.
   d. This is normal Web traffic

Correct answer: "b".

Bibliography for Part II

Calhoun, Chip. "Windows XP UPnP Exploits". Global Information Assurance Certification. 2 September 2002. 8 February 2003. <http://www.giac.org/practical/Chip_Calhoun_GCIH.doc>.

Cormier, André. "LOGS: GIAC GCIA Version 3.3 Practical Detect(s) (Andre Cormier)". 20 January 2003. intrusions@incidents.org. Archive by Universität Stuttgart's CERT. <http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00162.html>.

"DShield - Port Report for 137 - NETBIOS-NS". DShield.org. www.dshield.org. 25. January 2003. <http://www.dshield.org/port_report.php?port=137&Submit=>.

"Extreme Picture Finder registration". Extreme Internet Software site. 23 January 2003. 25 January 2003. <http://www.exisoftware.com/picture_finder/buy.html>.

Fyodor. "Nmap network security scanner man page". 16 September 2002. www.insecure.org. 25 January 2003. <http://www.nmap.org/nmap/data/nmap_manpage.html>.

Hassell, Riley, Marc Maiffret, Neothoth (pseudo.), and Ryan Permeh. "UPNP - Multiple Remote Windows XP/ME/98 Vulnerabilities". 20 December 2001. eEye Digital Security. 25 January 2003. <http://www.eeye.com/html/Research/Advisories/AD20011220.html>.

"NDHU Proxy Server (cache1) Statistics Page: Frequency of Access: Items". 27 November 2002. National Dong Hwa University. 25 January 2003. <http://cache3.ndhu.edu.tw/squid/days/requests/NDHU%20Proxy%20Server%20(cache1.ndhu.edu.tw).25.total-accesses.html>.

"Nmap - Free Stealth Port Scanner for Network Exploration & Security Audits. Runs on Linux/Windows/UNIX/Solaris/FreeBSD/OpenBSD". 10 August 2002. www.insecure.org. 25 January 2003. <http://www.insecure.org/nmap/>.

"Snort Report". 27 September 2002. University of Denver. 25 January 2003. <http://www.du.edu/~cgibbons/20020927.1.html>.

Thomas, Ashley. "Re: LOGS: GIAC GCIA Version 3.3 Practical Detect (Peter van Oosterom, fragmentation scanning)". 22 January 2003. intrusions@incidents.org. Archive by Universität Stuttgart's CERT. <http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00188.html>.

"Universal Plug and Play Membership List". UPnP Forum. Microsoft Corporation. 27 February 2003. <http://www.upnp.org/membership/members.asp>.

Part III -- Analyze This!

Management Summary

The analyst was asked to review five days' worth of logs for this university and provide an analysis of the data, that is, an estimate of the security status of the university. The analyst's conclusion is that the university has serious security problems that need to be addressed. Additionally, the university's intrusion detection system needs to be evaluated for its intended purpose. If it is intended to collect statistics and address the worst issues on campus, as may be a necessary approach in a university setting, the IDS is relatively well set up. If, however, it is the intent of the IDS to trigger action based on most alarms, the IDS needs much tuning.

This analysis begins by attempting to understand something of the network structure at the university, after which it presents a list of all of the alerts detected during the five day period, along with a frequency, explanation, and danger rating for each. After that, the alerts that earned a danger ranking of "high" are examined, with one intentional addition of a group of "medium" alerts. After the analysis, a list of recommendations to improve network security in the immediate future and in the long run is presented. Among the suggestions is a list of eight computers that were probably hacked and need to be attended to as soon as possible. At the very end, a loose description of the analyst's procedure for processing the data is included.

As the recommendations are primarily of a technical nature, they do not belong in a management summary. Management is encouraged to look at them, though. Perhaps most important among the recommendations is the long term, never ending goal of user

education. This is sometimes a forgotten battle, but it is one that we cannot afford to ignore.

File List

The following files were chosen for analysis:

- alert.030111, OOS_Report_2003_01_11_4183, scans.030111
- alert.030112, OOS_Report_2003_01_12_25129, scans.030112
- alert.030113, OOS_Report_2003_01_13_14787, scans.030113
- alert.030114, OOS_Report_2003_01_14_8867, scans.030114
- alert.030115, OOS_Report_2003_01_15_21827, scans.030115

Summary of alerts

Below is a list of all alerts generated in this five day period. They are ordered according to estimated danger level, then by frequency. Beneath the table is a small pie chart to give the reader a gut feeling of the proportions of alerts in all three danger levels. The pie chart is not intended to give the reader percents or other numbers. The alerts are presented here at the very beginning because this analysis will primarily be based on the alerts, and will use the out of spec and port scan information in a supporting fashion.

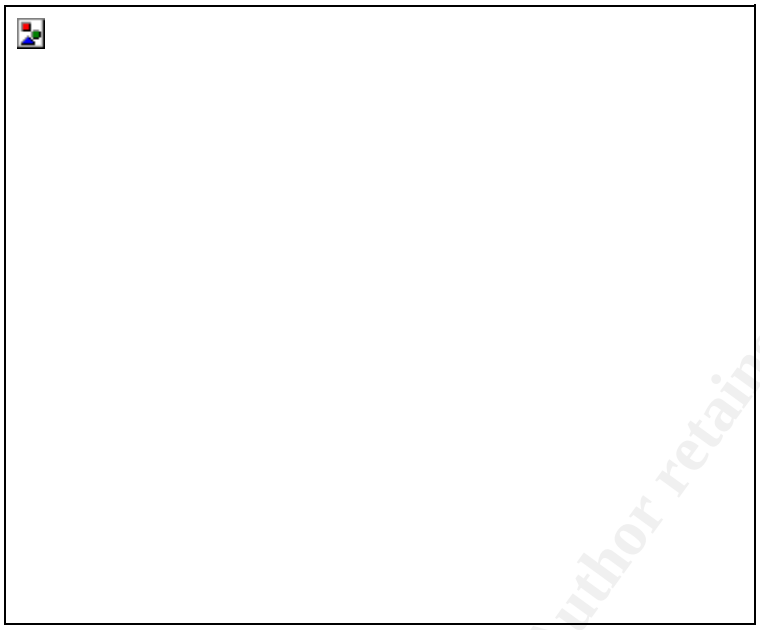
Count	Attack	Danger	Description
161605	High port 65535 tcp - possible Red Worm - traffic	High	This is a backdoor associated with the Adore worm.
24425	spp_http_decode: IIS Unicode attack detected	High	This is an attempt to use the Unicode encoding of URLs that lead to resources outside of the Web server's directory structure in order to bypass URL checking on the server side.
3607	High port 65535 udp - possible Red Worm - traffic	High	Again, this traffic is associated with the Adore worm.
2702	spp_http_decode: CGI Null Byte attack detected	High	CGI null byte attacks work by inserting a null byte into a command stream to bypass security checks on the server side that do not expect a null byte. (Urwiller)
2285	IDS552/web-iis_IIS ISAPI Overflow ida nosize	High	It has been claimed that this is an indication of either the Code Red II worm or the Nimda worm (Ellis), but this analyst has been unable to verify either assertion. The alert is not a standard Snort alert.
2036	SUNRPC highport access!	High	RPC implementations have had a number of vulnerabilities over the years, and this is an

Count	Attack	Danger	Description
			attempt to find a vulnerable RPC server.
284	Possible trojan server activity	High	This is set to trigger when the port 27374 is in use, because this port is used for the Windows trojan SubSeven.
131	SMB C access	High	This is attempted administrative access to an unprotected Windows share of the C drive. If read access is enabled, any files, including private ones, can be read. If write access is enabled, trojans could be planted.
7	RFB - Possible WinVNC - 010708-1	High	VNC is a free program that enables graphical console access over the network on both Unix and Windows.
7	Attempted Sun RPC high port access	High	See description above of a nearly identical alarm.
6	FTP passwd attempt	High	This is an attempt to transfer the system /etc/passwd file to the attacker.
2	NIMDA - Attempt to execute cmd from campus host	High	This is an attempt to propagate the Nimda worm by looking for a hole in Microsoft IIS that allows an attacker to execute cmd.exe.
2	EXPLOIT NTPDX buffer overflow	High	This is an attempt to exploit a buffer overflow in a daemon for the Network Time Protocol.
95848	Watchlist 000220 IL-ISDNNET-990517	Medium	These are presumably known hostile addresses that should be keep under observation.
17712	TFTP - External UDP connection to internal tftp server	Medium	These are attempts to find and contact internal Trivial FTP servers. These attacks can be important attacks, as TFTP requires no authentication to store and retrieve files. At the very least, an open TFTP server could become a warez server.
12042	TFTP - Internal TCP connection to external tftp server	Medium	These are more indicative of attempts by students to find open TFTP servers. These can also be attackers who have already compromised an internal machine and are now downloading tools.
2809	Watchlist 000222 NET-NCFC	Medium	These are also presumably known attackers that need to be kept under observation.
276	External RPC call	Medium	This is an attempt to contact the portmapper on the target host. This is an information gathering technique that is the prelude to an attack.
242	IRC evil - running	Medium	XDCC is a part of IRC that lets people share

Count	Attack	Danger	Description
	XDCC		files that others in the same channel can download. These are normally illegal copies of music or films. (TonikGin)
155	Port 55850 tcp - Possible myserver activity - ref. 010313-1	Medium	MyServer is a distributed denial of service tool that listens for commands, then attacks a specified target.
69	Port 55850 udp - Possible myserver activity - ref. 010313-1	Medium	MyServer is a distributed denial of service tool that listens for commands, then attacks a specified target.
4	TFTP - External TCP connection to internal tftp server	Medium	See descriptions of three similar attacks above.
2	EXPLOIT x86 NOPS	Medium	This is likely an alert that checks for the presence of many Intel "no operations", as described below in a similar alert. As such, it is less likely to be a false alarm.
1	Bugbear@MM virus in SMTP	Medium	Bugbear was an e-mail virus that won a certain amount of fame.
28610	SMB Name Wildcard	Low	Attempts to read the NetBIOS table of the attacked host. This is an information gathering attack that is especially useful against Windows domain controllers.
1715	EXPLOIT x86 NOOP	Low	These indicate that hexadecimal 90 was found in the payload, which is the "no operation" instruction on the Intel 32-bit x86 architecture. Exploit code often includes a number of these instructions. This is usually a false alarm.
1692	Queso fingerprint	Low	This is an attempt to remotely determine the operating system of the target host.
988	Null scan!	Low	A null scan is a TCP scan in which all TCP flags are off. It is for information gathering and firewall traversal.
162	TCP SRC and DST outside network	Low	This is most likely an indicator of spoofed packets originating from this network.
101	Tiny Fragments - Possible Hostile Activity	Low	A number of attacks and IDS evasion techniques are based on fragmenting packets, and there is a certain lower limit to the size of fragments that one would normally expect to see. Smaller packets are probably crafted for ill intent.
96	NMAP TCP ping!	Low	NMAP usually checks the host to be scanned

Count	Attack	Danger	Description
			to see if it's running, and if not it is not scanned. One method of checking is to send a crafted TCP packet with a unique signature. This is a prelude to an information gathering attack.
87	ICMP SRC and DST outside network	Low	Again, this is an indication of spoofed packets originating from the home network.
87	EXPLOIT x86 setuid 0	Low	This is once again a series of hexadecimal digits that on the Intel platform indicate an attempt to become the superuser. Usually a false alarm.
35	Incomplete Packet Fragments Discarded	Low	If a fragmented packet fails to be reassembled at the receiving end because one or more fragments were lost, the received fragments are discarded and an ICMP error is sent back. This can be used as a mapping technique.
27	EXPLOIT x86 setgid 0	Low	This is a string of hexadecimal digits that on Intel processors represent the machine code to set the group ID to the root/wheel group. Usually a false alarm.
18	TFTP - Internal UDP connection to external tftp server	Low	See descriptions of two similar attacks above.
13	MY.NET.30.4 activity	Low	Presumably this indicates activity from a host that should not be active.
9	External FTP to HelpDesk MY.NET.70.50	Low	Presumably this indicates an attempt to connect to an internal FTP server that is meant for use by the university help desk.
9	EXPLOIT x86 stealth noop	Low	This is another series of possible machine instructions for the Intel platform that perform the same "no operation" function, but are not as obvious to IDS sensors. Usually a false alarm.
8	Probable NMAP fingerprint attempt	Low	This is another attempt to identify the operating system of the targeted host.
7	SYN-FIN scan!	Low	This is an attempt at information gathering, operating system fingerprinting, or firewall traversal by setting both the SYN and the FIN flags in a TCP packet.
6	External FTP to HelpDesk MY.NET.70.49	Low	See description above for similar alert.

Count	Attack	Danger	Description
1	External FTP to HelpDesk MY.NET.83.197	Low	See description above for similar alert.



Relationships between machines in the traces

This is where we take a cursory look at the data, predominantly at the out of spec files, to discover any relationships that may exist between machines on the network, in the hopes that this information will help in the analysis phase later. The out of spec files were used because they are short enough that a semi-manual inspection is thinkable and may yield usable results, and because the out of spec files aren't good for much else. They mostly record packets that use ECN and scans that are based on abnormal TCP flags or something similar. The ECN logs are useless from an intrusion detection standpoint, and the scans are so unimportant next to what is really going on in this network as to be negligable.

Looking at the scans files, we find that the data come from UMBC. All scans either come from or are destined to 130.85.0.0/16, which is assigned to the University of Maryland Baltimore County.

Due to the overwhelming number of out of spec packets (1523) destined to MY.NET.6.40 on the SMTP port that were seemingly exclusively related to ECN, it is clear that MY.NET.6.40 is a mail server. There are other possible mail servers with much less traffic at MY.NET.139.230, MY.NET.24.22, MY.NET.145.9, MY.NET.140.236, MY.NET.179.77. This is based on the fact that there were also a few ECN packets destined to these addresses on port 25 in the out of spec files. MY.NET.145.9 also had the dubious honor of receiving the one instance of the Bugbear virus that was detected

during these five days. The question may remain: just detecting an ECN packet to port 25 doesn't guarantee that there is an e-mail server on the other end; is there better proof? Yes. The sheer number of packets to MY.NET.6.40 leaves no question as to its authenticity. Looking at the top initiators of traffic to the other suspected mail servers gives us vger.kernel.org for MY.NET.139.230, which is the mail server for the Linux kernel mailing list. Almost all of the rest of the senders are SMTP servers for either rapid-e.net or greatoffrs.com, which in the end are both the same thing, and are known spammers. (SBL5817) The entry in the ARIN database names a different company, presumably the company behind both rapid-e.net and greatoffrs.com:

```
OrgName:    Digital Connexxions
OrgID:      DICN

NetRange:   209.47.251.0 - 209.47.251.255
CIDR:       209.47.251.0/24
NetName:    DIGITALC-UUBLK1
NetHandle:  NET-209-47-251-0-1
Parent:     NET-209-47-0-0-1
NetType:    Reallocated
Comment:
RegDate:    2002-08-06
Updated:    2002-08-06

TechHandle: BT669-ARIN
TechName:   Taylor, Brock
TechPhone:  +1-905-338-8355
TechEmail:  btaylor@dconx.com
```

Finally, using the fact that we know this is UMBC's network, the addresses 130.85.6.40 and 130.85.24.22 resolve to mx4del.umbc.edu and mx2in.umbc.edu respectively. The others have oddball names.

There are many, many Web-enabled devices on campus. This is derived from the Web server attacks from the alerts logs. This analyst counted 777 separate Web servers of one form or another. Of the 777, only 630 could be resolved into names. An decent proportion of those are printers or copy machines. Using the search terms "print" (84), "hplj" (15), "xerox" (8), and "laserjet" (3), the analyst found 110 matches. Using the same method, there appear to be 4 webcams. Using the search term "pooled", the analyst discovered 92 Web-enabled machines that are probably part of campus computing pools for the students. Not all such computers seem to follow the same naming convention, meaning there are likely to be many more public systems that are running Web servers. These Web servers may be unintentional, or they may be there for administration purposes.

There are a number of TCP packets in the out of spec files that have no flags set. These are all destined to MY.NET.1.4 on port 37, which is for time synchronisation. All of these packets are from two internal sources (MY.NET.70.183 and MY.NET.53.10), which both turn out to be webcams. These are in all likelihood true time synchronisation

packets, since webcams usually print the date on the pictures they capture. The fact that they have no flags set may indicate a faulty TCP/IP implementation.

Based on the alerts concerning FTP traffic to the help desk, we can infer that MY.NET.70.50, MY.NET.70.49, and MY.NET.83.197 are computers for use by the university's help desk.

Although there are alerts for MY.NET.30.4, there is no information that would give us a hint as to this machine's function. All we know is that all of the alerts related to this machine were traffic to port 80 on MY.NET.30.4.

Analysis of attacks

All of the highly dangerous alerts are analyzed in depth, as well as two of the alerts rated as "medium" dangerous, namely the two watchlists. The watchlists are presumably in place because the originating networks are considered unfriendly and need to be kept under observation. The first watchlist in particular came in at second place in the frequency ratings, leading the analyst to suspect true foul play.

The huge number of alerts concerning the port 65535 is bad. This is the port often associated with the back door from the Adore worm (Rautiainen) Upon breaking the alarms out by address, we find that nearly 128000 of the 161000 alerts are associated with the internal address MY.NET.84.151. Further examination of these alerts yields the fact that all source ports are ephemeral when the destination port is 65535 to MY.NET.84.151. This machine has almost certainly been compromised. The next biggest originator of alerts on port 65535 is MY.NET.88.193, which generated a paltry 26000+ alerts. Analysis shows that these alerts are also associated with ephemeral source ports from the other side of the connection, leading us again to the conclusion that this machine has been compromised. The next most talkative host on port 65535 generated 33 alerts, meaning the other alerts were almost certainly false positives. Looking at it the other way, the attacker who targets the destination port 65535 more than any other attacker (22699 times) is 80.14.23.232, which resolves to APuteaux-105-1-5-232.abo.wanadoo.fr, and whose registration information is:

```
inetnum:      80.14.23.0 - 80.14.23.255
netname:      IP2000-ADSL-BAS
descr:        BSPUT105 Puteaux Bloc1
country:      FR
admin-c:      WITR1-RIPE
tech-c:       WITR1-RIPE
status:       ASSIGNED PA
remarks:      for hacking, spamming or security problems send mail to
remarks:      postmaster@wanadoo.fr AND abuse@wanadoo.fr
remarks:      for ANY problem send mail to
gestionip.ft@francetelecom.com
mnt-by:       FT-BRX
changed:      gestionip.ft@francetelecom.com 20020109
source:       RIPE
```

This is a DSL network for home users run by the French telecom.

The packets we just discussed are only packets destined to the university's network with a destination port 65535. There are also the internal machines that are connecting to external machines on port 65535. The top six offenders (chosen because they all generated more than 100 alerts) are: MY.NET.84.193 (3439 alerts), MY.NET.153.178 (646 alerts), MY.NET.150.213 (522 alerts), MY.NET.84.178 (389 alerts), MY.NET.70.176 (322 alerts) , and MY.NET.83.146 (287 alerts). All but MY.NET.70.176 (phaser.ucs.umbc.edu) appear to be public machines.

After reviewing some of the more recent analyses of this network, another analysis was found that also contained high amounts of port 65535 traffic, and that was the analysis authored by Steven Drew. (Drew) Mr. Drew provided lists of the most active hosts where this alert is concerned. None of the hosts in his lists match with the hosts in this analysis. Michael Holstein also detected this alert and did a brief analysis of it, suggesting that all machines that generated this alert be investigated immediately (Holstein). He included the entire list of potentially compromised machines, but even his very long list does not include the top generators of alerts found in these five days. It would seem that backdoors on the university network are a continuing problem that jumps from host to host.

The second most frequently occurring alert in the category "highly dangerous" is actually the collection of Web alerts, that is both of the spp_http_decode alerts and the IIS ISAPI alerts. Initially the analyst was not going to devote time to analyzing these attacks, going on the opinion that these attacks are high risk, but still noise. After all, every script kiddie tries a few Web attacks first, and thousands of alerts on Web attacks doesn't mean that a single machine was compromised. One would expect to see compromised machines as the sources for other alerts. The problem with this theory is twofold: 1) the attacks are still highly dangerous, and 2) in the case of worms, we would indeed expect to see compromised machines as the sources for other attacks, i.e. more Web attacks, which we have already chosen to ignore.

How to analyze these attacks presents a slightly different problem than the previous and following attacks. There are thousands of machines in these relationships, and the alerts are fairly well spread out across them, as opposed to the Adore alerts (port 65535), where only a couple of machines accounted for nearly all of the alerts. Our approach will be first to identify the top destination addresses for the sake of completeness, then to analyze the top source addresses. We will find that most of the source addresses (in contrast to the destination addresses) are internal hosts. In an attempt to determine if any of the hosts are compromised with a worm, we will look at the number of alerts from each machine plotted over time. If we can assume that all machines are left on all of the time, then we would expect to see an even distribution of attacks across time from a host infected by malware.

Only computers that were scanned at least 1000 times made the lists of the most scanned destination addresses and the heaviest scanning source addresses. The most scanned machines for Web attacks are:

1. 216.241.219.14 (scanned 1658 times)
2. 61.129.67.78 (scanned 1315 times)
3. 207.200.86.66 (myns-v2.websys.aol.com, scanned 1122 times)
4. 207.200.86.97 (myns-v1.websys.aol.com, scanned 1052 times)

Wait a minute. Not a one of those machines is on the university's network. That may mean the university is doing more scanning than it is being scanned. That's not a good sign.

The first address is registered to The Cobalt Group, Inc. in Seattle, Washington. Every one of these attacks comes from MY.NET.90.42, and, as we will see later, all Web attacks coming from MY.NET.90.42 are directed at this server. Calling up the IP address in a Web browser yields an apparently static error indicating that the browser being used (regardless of which browser is being used) does not meet the minimum requirements for the site. The viewer is instructed to download Netscape Navigator version 3 or higher, or Internet Explorer version 3 or higher. Connecting to the server with telnet to port 80 gives us the information that the server is running an ancient version of Apache (1.3.12) along with an ancient version of OpenSSL (0.9.5), and that there is a reverse proxy in front of it (NetCache NetApp/5.2.1R1). All of the attacks are CGI null byte attacks, to which any Web server can be susceptible. MY.NET.90.42 occurs not one other time in all of the files being analyzed. All of the attacks happen in the span of four minutes. Given that more than 800 attacks were attempted in one minute, this does not look much like a bunch of attacks. It looks much more like someone uploaded a large file to a CGI script. Perhaps the file was binary, and that triggered these alerts. These are most likely false alarms.

The second address belongs to Zhejiang Daily in China. Is this some kind of reverse attack against China after all of those worms and the generally negative sentiment against America? Not impossible. All of these attacks come from two hosts. The first host, which accounts for 1216 of the attacks, is MY.NET.144.61. The second host is MY.NET.84.218. According to UMBC's DNS, these are both public machines. Every single one of the attacks was an IIS Unicode attack. The purpose of trying this many times is not clear, unless every attempt was for a different Unicode-encoded file, or unless every attempt was a different Unicode encoding of the same thing. At the very least it is clearly active targeting. Or is it targeting at all? An examination of the attacks over time shows a clustering effect, much like Web surfing. The much more likely explanation is that, since this is a Chinese site, things are being Unicode encoded, as they should be, and that is setting off alarms in Snort's URL normalization plugin. This is simply someone reading a Chinese newspaper, and is a bunch of false alarms. Following is the graph of attacks against Zhejiang Daily over time from the most active attacking address.

The other most attacked hosts are both My Netscape servers. Once again, every single one of the attacks against the first server and almost all of the attacks against the second server come from one source IP address: MY.NET.85.74, and every last one is an IIS Unicode attack. Naturally, Netscape servers are running "Netscape-Enterprise/4.1", as returned by the server itself, and should not be susceptible to Unicode attacks. If these are attacks, the attacker has certainly not done her/his homework, and the tool being used isn't very intelligent, either. The timing reveals three clusters across three days, all around 12:00. Again, this looks more like normal Web surfing. Someone is probably sitting in a boring computer class and decided to surf instead of listening to the instructor. Either that, or it is someone's lunch break. Following is the graph of attacks against both My Netscape servers over time.

Finally we can take a look at the attacking hosts. The hosts that initiated more than 1000 attacks are:

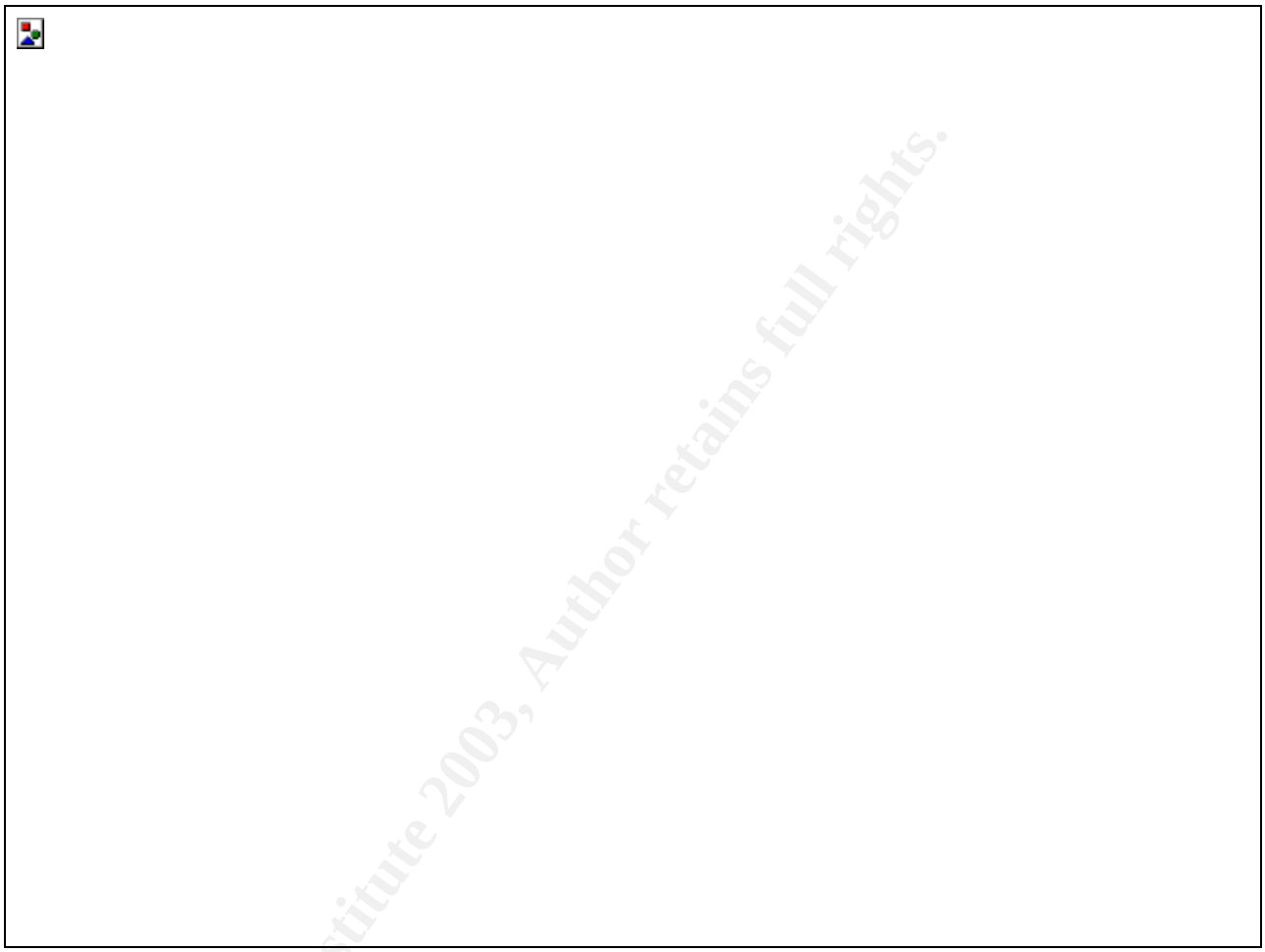
1. 148.246.52.7 (ce590-gdl02.terra.net.mx, 3802 attacks)
2. MY.NET.85.74 (2144 attacks)
3. MY.NET.90.42 (1658 attacks)
4. MY.NET.153.110 (1402 attacks)
5. MY.NET.144.61 (1234 attacks)

We have already seen our friends MY.NET.85.74, MY.NET.144.61, and MY.NET.90.42, and we know why they are on the list. ce590-gdl02.terra.net.mx belongs to TerraLycos Mexico and attacks a total of twenty different machines on the university's network. The attacks are well distributed across the victims, so we will resort to using timing to attempt to classify this traffic. The traffic arrives on two days: the 12th of January and the 15th of January. The graphs are below.

These graphs do not exhibit the kind of clustering we would normally associate with Web browsing. A count of the alerts from this address tells us that all of the alerts are IIS Unicode attacks. The out of spec files contain nothing from this address. The scans file is what really gives this attacker away. TerraLycos scans the university's network 4724 times, every one of which is to port 80. A total of 1792 separate IP addresses are scanned. The administrator of this computer needs to be contacted and informed that (s)he has a problem.

The only address that remains to be examined is MY.NET.153.110. It attacked a total of 30 different machines, almost all of which were from 210-218/8. This range of IP addresses is assigned to APNIC. Many of the attacks were to HANARO Telecom in Korea. Many belong to Daum Communication, also in Korea. This begins to look very much like the person who read Zhejiang Daily. Indeed, all of the attacks are Unicode attacks, and they are nicely clustered together on the time axis, as the following graph of the activity on the 12th of January shows. One of the Web servers that was attacked was checked, and it looks very much like a Korean portal along the lines of MSN or My Netscape. There were a huge number of graphics and other embedded items, which is surely the reason for the spike in the graph at the very beginning. It may also be that this is when the browser was called up on the client computer, and the browser is set to

automatically load a series of Korean Web pages in a sidebar. This would help to explain not only the spike at the beginning, but also the number of servers being attacked. After that, many things were cached by the browser. Still we see that it is very common to have between 50 and 100 objects being loaded in a minute. This was just someone surfing.



We begin to see that analyzing Web attacks leads to a lot of false positives. We will discuss possible ways to mitigate these false positives in the section for recommendations.

Looking again at the table of alerts, the second most frequently occurring alert overall is traffic from a network belonging to an Israeli ISP, whose RIPE record is listed here:

```
inetnum:      212.179.100.0 - 212.179.124.255
netname:      CABLES-CONNECTION
mnt-by:       INET-MGR
descr:        CABLES-CUSTOMERS-CONNECTION
country:      IL
admin-c:      MR916-RIPE
tech-c:       ZV140-RIPE
status:       ASSIGNED PA
```

```
remarks:      please send ABUSE complains to abuse@bezeqint.net
remarks:      INFRA-AW
notify:       hostmaster@bezeqint.net
changed:      hostmaster@bezeqint.net 20021029
source:       RIPE

route:        212.179.64.0/18
descr:        ISDN Net Ltd.
origin:       AS8551
notify:       hostmaster@bezeqint.net
mnt-by:       AS8551-MNT
changed:      hostmaster@bezeqint.net 20020618
source:       RIPE
```

Why are there so many alerts for this network? About two thirds of them (68218) come from seemingly one connection with MY.NET.150.5. The connection took place on the 13th of January between 8:58 and 10:39 in the morning. The average packet arrival rate during that period of time was a little more than eleven packets per second. The transfer had two peaks, one at 9:17, and the other at 9:37. At both peaks there are more than 23 packets per second arriving. The following graph depicts the number of packets received by MY.NET.150.5 during the transfer per minute.

The analyst's first assumption was that this was a denial of service attack. Upon closer inspection, we see that this is relatively unlikely. In the end, there probably aren't enough packets to perform a true denial of service. The packets all come from the same source address and port, meaning a normal TCP denial of service is not at work.

Looking at the graph, we see three spots each 3-4 minutes long where no packets are sent. One possibility is that the packets were coming so fast that the IDS overwrote the log buffer and made those lines in the log unintelligible, as is often the case in the alerts files. Upon further investigation, it seems that there truly were no arriving packets during those times. Those times were 9:41-9:44, 9:57-9:59, and 10:27-10:29. These could well correspond to pauses between file transfers. There appears to be no other communication between the two hosts in question on that day.

The ports 1540 and 3321, the destination and source ports in this communication, respectively, don't seem to have any normal use. Port 1540 is registered as rds, and port 3321 is registered as vnsstr. There do not appear to be any trojans, viruses, backdoors, or other such things commonly associated with either. (Tracker, "Ports and ...")

Assuming Ethernet as a transport medium and full data packets, we only come to a maximum throughput of about 275 kbps. This truly does seem to be a file transfer (or 4) of some kind. What was being transferred, and why are those specific ports being used? One obvious possibility is passive FTP, but, as already noted, there was no other communication between those two hosts on that day. This leads us to the distinct possibility that there is a backdoor on one of the two machines. Without more information, it is impossible to say on which machine or what may have been transferred.

Only about 27000 packets remain unaccounted for from the Israeli network. Another 16332 are directed at the host MY.NET.113.4 on port 1214, which is the standard KaZaa port. MY.NET.113.4 resolves to ucommons-113-04.pooled.umbc.edu, leaving us to believe this is probably in a common computer lab for students. Our suspicions are confirmed after a quick look at the UMBC map, which pinpoints the location of "The Commons". ("CAMPUS MAP") Looking at the other buildings in the surrounding area leads us to the conclusion that this part of campus is not for student housing, but rather for administration and classes. The fact that a common computer seems to be hosting a KaZaa server is disturbing. Everyone knows that KaZaa is mostly used for trading illegal copies of music and films. According to UMBC's acceptable use policy, all computer users shall "[r]espect all pertinent licenses, copyrights, contracts, and other restricted or proprietary information." (Office of Information Technology)

Of the remaining 11000 packets from the Israeli ISP, about 9000 come from the source port 80, meaning they are likely to be responses from Web servers in that network. The university's users provided the stimulus, and these are not likely to be attacks.

An additional 800 to 900 packets are destined to MY.NET.91.252 on port 1237. Port 1237 is officially assigned to tsdos390, but is also used for the game Red Alert 2. (Brentano) Let's look at some statistics concerning packet frequency to decide if this

looks like gaming traffic or something else. The first three days (and the first meaningful minute of the fourth day) are best presented in text format. There are two columns: the number of occurrences, and the time.

```
  2 01/11-00:54
 13 01/11-07:36
524 01/11-08:00
175 01/11-08:01
  2 01/11-08:24
  8 01/12-21:24
 11 01/12-21:25
  1 01/12-21:27
  1 01/13-08:42
  3 01/14-00:46
```

So far, this hardly looks like gaming traffic. It looks like one big transfer and some garbage. Not knowing the game Red Alert 2, it is reasonable to suppose that the game requires a certain amount of setup when a new user joins. This may be what we are seeing on the 11th at 8:00. The 14th begins to look like one would expect for gaming. The only question that remains in this analyst's mind is if this is enough traffic for gaming. Again, having never played online games, it's hard to say. The graph of traffic on the 14th in ten minute resolution follows.

The remaining 1000 packets or so are hardly worth trying to analyze. Some of them are obviously KaZaa or scans for mail servers or Web servers, and some of them use strange source and destination ports, and would be difficult to analyze without the data in the packet.

Michael Holstein, Joe Ellis, and Rick Yuen all analyzed this traffic in their practicals, but all discovered it was primarily Web and KaZaa traffic. (Holstein, Ellis, Yuen)

There is another network on the watchlist, and as it is on the watchlist, it is safe to assume that traffic to and from that network needs to be monitored. Based on one of the addresses that triggered this alert, these packets come from the Computer Network Center Chinese Academy of Sciences, whose information is as follows:

```
OrgName:    The Computer Network Center Chinese Academy of Sciences
OrgID:      CNCCAS

NetRange:   159.226.0.0 - 159.226.255.255
CIDR:       159.226.0.0/16
NetName:    NCFC
NetHandle:  NET-159-226-0-0-1
```

```
Parent:     NET-159-0-0-0-0
NetType:    Direct Assignment
NameServer: NS.CNC.AC.CN
NameServer: GINGKO.ICT.AC.CN
Comment:    The information for POC handle QH3-ARIN has been reported
to
            be invalid. ARIN has attempted to obtain updated data, but
has
            been unsuccessful. To provide current contact information,
            please email hostmaster@arin.net.
RegDate:    1992-06-11
Updated:    2002-10-08

TechHandle: QH3-ARIN
TechName:   Xiqiong, Zhang
TechPhone:  10 82616000
TechEmail:  zxq@cstnet.net.cn
```

Given that it's a class B network, there aren't all that many alerts pertaining to it. This is also an academic institution, and it would hardly be surprising to discover that people from UMBC and from this Chinese Academy of Sciences are working together. Looking at a link graph depicting the source and destination ports involved in all of these alerts will help us.

From this we can see that the vast majority of the traffic was generated as a response to Web pages that our users called up. There is also some mail traffic, which would tend to support the assertion that there is collaboration between the two institutes of learning, especially since the amount of mail is rather low. 140 packets could be a single mail with a medium sized attachment. We also see the associated ident traffic, which would more tend to indicate that 5 mails were sent, and that they were sent from the Chinese Academy of Sciences to UMBC. Besides that, we see a little FTP going on. Given the amount of FTP, it is also safe to assume that this is at least not any kind of warez traffic, though evil intent is still not impossible. Evil intent with an FTP server is best picked up by other IDS rules. Finally, we have connections between high ports on both sides. A quick visual inspection of some of the relevant connection information (listed below) makes it plain that this is exclusively passive FTP.

List of ports in high port to high port connections and related FTP connections:
63504 -> 21
63505 -> 21
63506 -> 4656
63507 -> 4655

63508 -> 21
63511 -> 4657

Once again, Michael Holstein found this traffic and analyzed it briefly, but he discovered traffic that may have been malicious. Specifically, he discovered traffic on ports 1752 and 1753. He does not indicate in which direction this traffic flows, or if there may have been something else that stimulated this traffic, like a passive FTP transfer.

Most of the RPC alerts (not including the "External RPC call" alerts) are false alarms. They trigger on the port 32771, which is often used for RPC services. It's also an extremely common ephemeral port. After filtering out Web and e-mail traffic, we found an SSH session between dromio.nist.gov and zeus.engr.umbc.edu, which we considered very interesting, but most likely legitimate. It also became obvious that someone is synchronizing their clock with NIST: every twelve hours for three twelve hour periods a reply came back from time-a.nist.gov from port 37 (for time synchronization).

Grand total, there were 15 attackers on port 32771 and 47 defenders. The top three defenders deserve to be looked at to determine if they have been compromised. Sliding into spot number one on the hit charts is MY.NET.132.50 with 12 attacks against it. Looking at the other log entries for this machine, we find the usual plethora of SMB wildcard attacks in addition to the RPC attacks, which all (but one) come at the same time from the same address (207.46.106.127/baym-cs127.msgr.hotmail.com). But wait! Right there is our indication that these are false alarms. The "attacking" address belongs to Microsoft, and upon closer inspection, these packets come from port 1863, which is used for the MSN Messenger Protocol. Thank goodness! What about the next one? MY.NET.55.66 has exactly the same story to tell. Looking at the alerts (reordered to be sequential through time, which they are not in the log file), we discover that the packets come with almost extreme regularity, leading us to believe that this is not attack:

```
01/14-10:55:37.557319  [**] SUNRPC highport access! [**]
64.12.29.129:5190 -> MY.NET.55.66:32771
01/14-10:55:39.725104  [**] SUNRPC highport access! [**]
64.12.29.129:5190 -> MY.NET.55.66:32771
01/14-10:56:39.350846  [**] SUNRPC highport access! [**]
64.12.29.129:5190 -> MY.NET.55.66:32771
01/14-10:57:39.344349  [**] SUNRPC highport access! [**]
64.12.29.129:5190 -> MY.NET.55.66:32771
01/14-10:59:39.359764  [**] SUNRPC highport access! [**]
64.12.29.129:5190 -> MY.NET.55.66:32771
01/14-11:02:39.389350  [**] SUNRPC highport access! [**]
64.12.29.129:5190 -> MY.NET.55.66:32771
01/14-11:06:39.472875  [**] SUNRPC highport access! [**]
64.12.29.129:5190 -> MY.NET.55.66:32771
01/14-11:09:12.014549  [**] SUNRPC highport access! [**]
64.12.29.129:5190 -> MY.NET.55.66:32771
01/14-11:10:39.534189  [**] SUNRPC highport access! [**]
64.12.29.129:5190 -> MY.NET.55.66:32771
01/14-11:11:39.534691  [**] SUNRPC highport access! [**]
64.12.29.129:5190 -> MY.NET.55.66:32771
```

```
01/14-11:16:39.526431  [**] SUNRPC highport access! [**]
64.12.29.129:5190 -> MY.NET.55.66:32771
```

Port 5190 is used for AOL Instant Messenger, and the address 64.12.29.129 belongs to America Online, Inc. Another false alarm.

Now for number three. MY.NET.87.126 is "attacked" 7 times, all from port 21, all from the same IP address (216.228.115.20), and all within a relatively short period of time (a little over one hour). While it's possible that the attacker is using a source port of 21 to evade packet filters, the far more likely explanation is that someone is downloading drivers for his/her video card: the attacking IP address belongs to Nvidia.

While we're on the subject of false alarms triggered by static searches for ephemeral ports commonly used for evil purposes, let's move to the related topic of SubSeven. All of the alerts labeled "Possible trojan server activity" appear because the port 27374 is in use. This port is typically associated with the SubSeven server, among a whole host of other trojans and worms (Ramen and Lion prominent among them). After filtering out the obvious Web and KaZaa traffic and only looking at packets coming from the university's network, we are left with six alerts:

```
01/14-15:54:46.081664  [**] Possible trojan server activity [**]
MY.NET.137.1:27374 -> 80.198.32.196:3114
01/14-15:54:57.190765  [**] Possible trojan server activity [**]
MY.NET.137.35:27374 -> 80.198.32.196:2360
01/14-15:54:58.291937  [**] Possible trojan server activity [**]
MY.NET.137.38:27374 -> 80.198.32.196:2651
01/14-15:54:58.884965  [**] Possible trojan server activity [**]
MY.NET.137.38:27374 -> 80.198.32.196:2651
01/15-13:49:15.421401  [**] Possible trojan server activity [**]
MY.NET.137.1:27374 -> 24.198.150.248:4059
01/15-13:49:23.749691  [**] Possible trojan server activity [**]
MY.NET.137.17:27374 -> 24.198.150.248:4620
```

These should represent the machines that replied to scans for the SubSeven backdoor. It appears that MY.NET.137.1, MY.NET.137.35, MY.NET.137.38, and MY.NET.137.17 all have SubSeven installed and need to be cleaned up. MY.NET.137.38 in particular sent two packets to the attacker in close succession, indicating an even higher probability that it is not clean. The only one in the list that has a name in DNS is MY.NET.137.1, which is named ernie.umbc.edu. The attackers are 0x50c620c4.abnxx5.adsl-dhcp.tele.dk and ptd-24-198-150-248.maine.rr.com, standard home users. Mr. Holstein also found this activity, identified it as the Ramen worm, but, again, none of the potentially compromised hosts he found match the hosts found in these five days. The same is true of the machines that Steven Drew and Tod Beardsley found; their lists are completely different.

The SMB C access alerts come from 85 hosts and go to a total of 15 hosts, those being:

1. MY.NET.190.102 (31 attacks)

2. MY.NET.190.100 (22 attacks)
3. MY.NET.137.34 (12 attacks)
4. MY.NET.137.46 (11 attacks)
5. MY.NET.132.43 (10 attacks)
6. MY.NET.190.38 (9 attacks)
7. MY.NET.190.17 (9 attacks)
8. MY.NET.137.35 (7 attacks)
9. MY.NET.190.90 (5 attacks)
10. MY.NET.132.24 (4 attacks)
11. MY.NET.132.45 (3 attacks)
12. MY.NET.132.42 (3 attacks)
13. MY.NET.190.34 (2 attacks)
14. MY.NET.132.22 (2 attacks)
15. MY.NET.132.27 (1 attack)

Looking for any other alerts concerning these addresses leads us to a dead end. All of the addresses experienced SMB wildcard scanning, sometimes in concert with the access to the C drive, sometimes not. The address MY.NET.137.46 experienced one more alert for SubSeven activity, but that was outgoing and unrelated to the SMB scans. MY.NET.137.35 also has some potential SubSeven activity connected to it, as we just saw. MY.NET.190.90 enjoyed himself scanning about 500 other machines for open shares (ports 137 and 139), but was otherwise harmless. MY.NET.132.42 experienced another 8 Web attacks (all seemingly unsuccessful) from three different sources. All in all, despite the potential danger in allowing anyone access (and possibly write access) to the C drive of a Windows machine, these turned out to be blissfully boring alerts.

No other analyses of the university's network included an analysis of these alerts, but this isn't really necessary. DShield tells us that this port is constantly scanned. ("DShield ...", port 137)

WinVNC (or just VNC), for the uninitiated, is a remote console program for Windows and Unix. It normally runs on port 5900. The alerts seem to target responses from VNC to outside addresses, and using more than just port numbers. VNC is not encrypted, and sends certain parameters upon connection establishment. In particular, it seems to be able to negotiate the Remote Framebuffer Protocol. This rule must certainly check for an RFB string in the response. In fact, the standard Snort rulebase includes a very similar rule:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"POLICY VNC server
response"; flow:established; content:"RFB 0"; offset:0; depth:5;
content:".0"; offset:7; depth:2; classtype:misc-activity; sid:560;
rev:5;)
```

The interesting thing about these scans is not just that they have the potential to be very malicious. They are also an indication of active targeting. Let's look at the alarms briefly:

```
01/11-14:54:24.165236  [**] RFB - Possible WinVNC - 010708-1 [**]
MY.NET.113.63:5900 -> 68.50.80.108:62482
01/11-14:54:37.453254  [**] RFB - Possible WinVNC - 010708-1 [**]
MY.NET.113.63:5900 -> 68.50.80.108:62483
01/12-12:16:31.200946  [**] RFB - Possible WinVNC - 010708-1 [**]
MY.NET.113.63:5900 -> 68.50.80.108:65459
01/12-16:23:21.795353  [**] RFB - Possible WinVNC - 010708-1 [**]
MY.NET.130.64:5900 -> 151.196.124.145:4738
01/14-23:37:04.925077  [**] RFB - Possible WinVNC - 010708-1 [**]
MY.NET.113.66:5900 -> 68.50.80.108:61960
01/14-23:37:17.968674  [**] RFB - Possible WinVNC - 010708-1 [**]
MY.NET.113.66:5900 -> 68.50.80.108:61961
01/14-23:37:35.924696  [**] RFB - Possible WinVNC - 010708-1 [**]
MY.NET.113.66:5900 -> 68.50.80.108:61962
```

Searching for these IP addresses in the port scan and alerts logs leaves us empty-handed, so they were not scanning the entire subnet for VNC servers. They knew where the VNC servers are. The IP addresses are suspicious. The one that connects the most is from a cable modem customer in New Jersey and is almost certainly a hacker. That address appears the most often, and there are multiple attempts to connect in quick succession, probably meaning that the attacker is guessing passwords. The other address is a Verizon DSL customer in the Virginia/D.C./Maryland area. This is also only one connect over the five days. This may well be an administrator connecting from home. If this doesn't appear again, it shouldn't be fretted over. The targets, by the way, are all kiosks somewhere in the University Commons, if the names in DNS can be trusted.

Once again, Mr. Holstein found the same alert in his analysis, but the attacked machines he found are not the same as the ones listed here, and he did not give the IP addresses of the attacking machines. (Holstein)

The FTP passwd attempts come from exactly two sources: 195.242.116.68 (dyn-195-242-116-68.ppp.tiscali.fr) and 212.194.157.77 (f08v-2-77.d1.club-internet.fr). The first address never appears again. The second address is involved in heavy SMB scanning. There are more than 700 alerts generated for this source address concerning SMB wildcard scans. Looking at the port scan logs, we find that this attacker generated 2276 entries, almost exclusively for ports 137, 139, and 445, that is, the SMB ports (old and new). The machines attacked have mostly high profile names (asp1.umbc.edu, bookstore.umbc.edu, dragon5.ifsm.umbc.edu, and one didn't resolve). asp1 was hit by 327 IIS attacks, but fails to turn up elsewhere in the logs. bookstore was attacked with IIS attacks 130 times, had one rather uninteresting tiny fragmentation attack, and the rest of the alerts were one transfer wherein the client happened to use the port 27374. dragon5 experienced one, single, lonely IIS attack. The last machine had no additional entries in the alerts files. This is all very good to know so we know how widely these attackers are targeting the university's machines. This attack, however, is an attempt to transfer the system password file to the attacker's machine so the attacker can set a password cracker against it. There is no way for us to know if the attacker succeeded based only on these alerts. The best course of action is to change all passwords on the attacked systems if the systems don't use some kind of a shadow password scheme. The machines attacked are MY.NET.5.92, MY.NET.113.208, MY.NET.130.187, and MY.NET.157.105.

No correlations were found for this attack.

Analysis of the two NIMDA alerts is surprising in the sense that they turn out to be false alarms. There are only two alerts, and searching for the originating hosts in all of the other log files yields exactly one further alert:

```
01/13-00:28:02.705108  [**] EXPLOIT x86 NOOP [**] 207.46.131.197:80 ->
MY.NET.112.223:1080
```

This alert was twenty minutes after the supposed attack from the campus host. The port 1080, normally for the SOCKS proxy, is a coincidence. The source address resolves to three different names, all under windowsupdate.com or microsoft.com, which would obviously be delivering objects with x86 noops. These both appear to be false alarms, though a false alarm on NIMDA somewhat surprises this analyst. It is even encouraging to see someone downloading patches from Microsoft.

The NTPDX overflow attempts are odd ones. First of all, these attacks aren't seen very often. Second of all, both of the target computers appear to be public machines (based on the names in DNS). Third of all, there is no indication that one of the attackers targeted any other machines on the network with any exploit at all (the address never appears again in the alerts or port scan logs). The other address has another story to tell. The alerts are:

```
01/13-14:22:51.447252  [**] EXPLOIT NTPDX buffer overflow [**]
64.7.192.173:16293 -> MY.NET.88.164:123
01/15-17:54:13.905154  [**] EXPLOIT NTPDX buffer overflow [**]
211.106.66.141:54661 -> MY.NET.53.41:123
```

64.7.192.173 is registered to SiteStream, Inc. in California, and has the name media-01-002.publichost.com. SiteStream (www.sitestream.com, also www.publichost.com) specializes in streaming media. Let's take a look at all of the alerts from this address:

```
01/13-07:25:07.583261  [**] High port 65535 udp - possible Red Worm -
traffic [**] 64.7.192.173:6257 -> MY.NET.88.164:65535
01/13-09:52:01.388198  [**] High port 65535 udp - possible Red Worm -
traffic [**] 64.7.192.173:65535 -> MY.NET.88.164:65280
01/13-11:15:32.014096  [**] TFTP - Internal UDP connection to external
tftp server [**] 64.7.192.173:69 -> MY.NET.88.164:32177
01/13-14:22:51.447252  [**] EXPLOIT NTPDX buffer overflow [**]
64.7.192.173:16293 -> MY.NET.88.164:123
01/14-10:16:54.130756  [**] High port 65535 udp - possible Red Worm -
traffic [**] 64.7.192.173:30761 -> MY.NET.88.164:65535
```

This looks very much like a break-in. First we have the attacker connecting to a host with a backdoor on port 65535, probably sending a command or series of commands in one UDP packet. Perhaps those commands told the computer on the campus network to listen on port 65280 -- this port is also unusually high -- and the attacker then transferred a

small file to the internal host. Next we have what appears to be TFTP. The alert is set to trigger on a response to reduce the number of false alarms. Why the NTP exploit is tried against the internal machine is not clear, unless the attacker wanted to test it out before using it on other machines. This all takes place in the course of a day. The next day the attacker once again connects to the compromised machine for an unknown reason. Nothing happens on the last day of logs. As already stated, this series of alerts needs to be investigated immediately, because a compromised machine is almost certain. Why did this host not turn up in the analysis of the alerts about port 65535? Because only three alerts for this machine appeared over the course of five days, and those were deemed to be scans or false alarms. That just goes to show that an analyst has to have the time to investigate the less important or less pressing or more interesting alerts, because they can lead to unexpected insight into intrusions. One more question: is SiteStream, Inc. the attacker? Probably not. While it's certainly possible that an employee there spends his/her time hacking other computers, the more likely explanation is that SiteStream, Inc. has itself been hacked and is being used as a cover for the attacker's further activity. The attack times tend to confirm this: the times are probably East Coast, and California would be a few hours behind that, meaning someone was at work in the wee hours of the morning attacking the university. One would have to contact SiteStream, Inc. to find out for sure.

Correlations from GCIA practical analyses were not found. The alert appears in a few of the analyses, but no one analyzes them. The source and destination addresses do not appear in any of the examined analyses. DShield reports the source address as having twelve records against it, all directed at one host. Hardly newsworthy. DShield also reports that NTP is attacked a few thousand times a day, but that, too, is not exciting.

Examining the "EXPLOIT x86 NOPS" alarms doesn't seem to bring us much, but it was done out of curiosity:

```
01/15-06:39:11.331010  [**] EXPLOIT x86 NOPS [**] 80.200.225.161:1025 -
> MY.NET.84.151:1
01/15-14:11:50.420425  [**] EXPLOIT x86 NOPS [**] 80.200.225.161:1025 -
> MY.NET.84.151:1
```

We know the target machine because it was the top target for Adore-related packets. We will find out later that the attacker is also in the top ten list of attackers who executed dangerous attacks against the university. Knowing this, it would be good to examine all traffic (if possible) from this source. We have to ignore Adore-related traffic, because there is just too much of it. The other alerts concerning this address are:

```
01/15-06:39:05.801607  [**] SUNRPC highport access! [**]
80.200.225.161:1025 -> MY.NET.84.151:32771
01/15-06:39:11.296101  [**] Probable NMAP fingerprint attempt [**]
80.200.225.161:1027 -> MY.NET.84.151:135
01/15-06:39:11.299813  [**] NMAP TCP ping! [**] 80.200.225.161:1028 ->
MY.NET.84.151:135
```

```
01/15-06:39:11.308896  [**] NMAP TCP ping! [**] 80.200.225.161:1026 ->
MY.NET.84.151:1
01/15-06:39:11.331010  [**] EXPLOIT x86 NOPS [**] 80.200.225.161:1025 -
> MY.NET.84.151:1
01/15-06:42:59.379111  [**] TFTP - External TCP connection to internal
tftp server [**] 80.200.225.161:1207 -> MY.NET.84.151:69
01/15-06:42:59.379287  [**] TFTP - External TCP connection to internal
tftp server [**] MY.NET.84.151:69 -> 80.200.225.161:1207
01/15-14:11:50.376130  [**] Null scan! [**] 80.200.225.161:1026 ->
MY.NET.84.151:135
01/15-14:11:50.420425  [**] EXPLOIT x86 NOPS [**] 80.200.225.161:1025 -
> MY.NET.84.151:1
01/15-14:12:12.934264  [**] Probable NMAP fingerprint attempt [**]
80.200.225.161:1028 -> MY.NET.84.151:135
```

In other words, the attacker pounds away at MY.NET.84.151, but doesn't seem to get anywhere. Most of the attacks are information gathering attempts (the nmap scans), but we see also that the attacker tries connecting to RPC ports and a TFTP server. The response from the TFTP server is probably a RST (we're dealing with TCP here). Are we sure? Let's lean some more on the port scan and out of spec files for help. As it turns out, the pause evident in the alerts file between 6:39 and 6:43 is due to an nmap scan. Port 69 was just one of the many ports, and happened to trigger some rules on the IDS. There are a total of 1433 entries in the corresponding port scan file, but no entries in the out of spec file during this time period. It looks like a completely normal nmap scan with OS fingerprinting turned on, as also indicated from the alerts. Is this dangerous activity? No. The university's defenses held under this prolonged attack. It's just fun to look at a hacker bang her/his head against a wall.

DShield has no records for the IP address of the attacker.

Looking now at the port scans that are taking place during the analysis period, we can certainly all guess what appears at spot #1: file sharing. In fact, both in terms of source ports and in terms of destination ports, WinMX (a common file sharing application) in particular appears far more often than anything else. The next closest port under destination ports (445 for SMB directly over TCP) appears less than one tenth as often as the WinMX port (6257). ("Internet Storm Center ...")

As mentioned, port 445 is in second place, and port 80 appears about half as often, putting it in third place. Fourth place is interesting, port 27005. This port is registered to FlexLM, which is a license manager used by many software applications. As normal as it would be to find a high incidence of this port in a campus setting, there is another reason why this appears. Looking more closely at the scans, we find that none of them are querying internal license servers, but rather are scanning huge numbers of external machines. Port 27005 is used for the game Half-Life. (watford(hi2u)@uiuc.edu) Why people are scanning for port 27005 is a mystery to this analyst, since that port is supposed to be for the Half-Life client, but there's certainly a good reason for that. The analyst has never played Half-Life.

One really has to wonder if the students have nothing better to do with their time.

Next in the list of the top destination ports for port scans is 135, which is normally for DCE-RPC. DCE-RPC is used most often in Outlook to Exchange server connections. The list of scanners is short: only 19 for a total of 43315 scanned machines. The attackers are all external except for one attacker, who comes in at place number 2: rwd-239.umbc.edu. The top scanner for port 135 comes from the University of Arkansas: 150.208.86.124, which does not resolve to a name. The registration information follows:

```
OrgName:    University of Arkansas
OrgID:      UNIVER-249

NetRange:   150.208.0.0 - 150.208.255.255
CIDR:       150.208.0.0/16
NetName:    ARKNET
NetHandle:  NET-150-208-0-0-1
Parent:     NET-150-0-0-0-0
NetType:    Direct Assignment
NameServer: DNS2.STATE.AR.US
NameServer: DNS1.STATE.AR.US
NameServer: DNS3.STATE.AR.US
Comment:
RegDate:    1991-06-11
Updated:    2002-07-23

TechHandle: ZS25-ARIN
TechName:   State of Arkansas
TechPhone:  +1-501-682-0500
TechEmail:  hostmaster@dcs.state.ar.us
```

Once again, DShield has no records for this IP address in its database.

The last recognizable ports are 21, 1214 for KaZaa, 3389 for Microsoft Terminal Services or Citrix Metaframe, and 443.

Next comes a brief analysis of the most scanned internal machines. Topping the list is an old friend: 130.85.84.151. This one has already been shown to have fallen prey to the Adore worm. Only four internal machines were scanned more than 1000 times. The remaining three are 130.85.82.248, 130.85.88.242, and 130.85.53.51. All three are public machines.

The last bit of information to be gleaned from the port scan logs that can be of some practical interest to us is the list of internal machines initiating scans. We will differentiate between this list including the file sharing/gaming users and without the file sharing/gaming users. File sharing is a policy concern and can also be used to propagate malware, but other scans are more interesting from a technical security point of view. Gaming is usually neither a policy problem nor a technical security problem of any great measure. It's just a waste of bandwidth. The list including file sharing users has eight machines that scanned more than 100000 targets. The list is:

1. 130.85.70.176 (phaser.ucs.umbc.edu)

2. 130.85.83.146 (aciv-83-146.pooled.umbc.edu)
3. 130.85.91.252 (fh-91-252.pooled.umbc.edu)
4. 130.85.84.178 (engr-84-178.pooled.umbc.edu)
5. 130.85.150.213 (libpc11.lib.umbc.edu)
6. 130.85.87.50 (chem-87-50.pooled.umbc.edu)
7. 130.85.162.90 (sparklin-1.umbc.edu)
8. 130.85.150.215 (libpc13.lib.umbc.edu)

We have already seen the top two in other places. They both made it as top machines looking for the backdoor installed by the Adore worm on port 65535. It's time to hunt those machines down and take a look at them. The same can be said of 130.85.84.178 and 130.85.150.213, which also made both lists.

The list of internal hosts initiating scans discounting file sharing and gaming is shorter (the cutoff is still 100000 scanning targets), but the faces are all familiar:

1. 130.85.91.252 (fh-91-252.pooled.umbc.edu)
2. 130.85.150.215 (libpc13.lib.umbc.edu)
3. 130.85.162.90 (sparklin-1.umbc.edu)

As much as the users of the machines in the first list may need a slap on the wrist, these three machines have a much higher potential of harboring malice. They should be dealt with accordingly.

There is one problem with investigating public machines, and that is determining if the machine is cracked or if the activity was the result of a logged in user who executed scans. This problem is necessarily left up to the local administrators.

Finally, it is of interest to take the alerts listed as "high" in the "danger" column and find out who has been trying these attacks against the university. This criterion is chosen because these are the most likely to be truly dangerous attackers. The top ten list of dangerous attackers, along with corresponding hostnames and the number of serious attacks executed against the university is:

1. 80.14.23.232 (APuteaux-105-1-5-232.abo.wanadoo.fr), 22681 attacks
2. 80.200.150.161 (161.150-200-80.adsl.skynet.be), 9952 attacks
3. 217.136.72.253 (253.72-136-217.adsl.skynet.be), 9779 attacks
4. 80.200.225.161 (161.225-200-80.adsl.skynet.be), 5908 attacks
5. 148.246.52.7 (ce590-gdl02.terra.net.mx), 3802 attacks
6. 80.11.172.105 (APuteaux-104-1-4-105.abo.wanadoo.fr), 2587 attacks
7. 217.136.71.2 (2.71-136-217.adsl.skynet.be), 2379 attacks
8. 80.13.100.3 (AClermont-Ferrand-201-1-2-3.abo.wanadoo.fr), 2278 attacks
9. 24.116.131.218 (24-116-131-218.cpe.cableone.net), 2273 attacks
10. 81.50.44.197 (AClermont-Ferrand-201-1-5-197.abo.wandoo.fr), 1811 attacks

It looks like abo.wandoo.fr and skynet.be are competing to be on the university's next watch list. That may well be a good place to put them.

Checking these IP addresses against the DShield database with their online IP report yields absolutely no records for some of the addresses, and one to two digit numbers of entries for the others. Certainly not what we would expect from the top attackers of a university. It is obvious from this alone that UMBC does not take part in DShield.

Recommendations

Having demonstrated that there are serious security problems in the university's network, it remains to describe a few ideas to improve security now and in the long run. The recommendations are not in any particular order. They all need to be addressed, so ordering is irrelevant.

As mentioned in the summary above, the IDS is well implemented if the intent is to collect statistics, find the worst offenders, and have a good idea of the big trends on the campus network. If the intent is that most alarms be acted upon, many of the alarms need to be deactivated, or the alarm criteria need to be made more specific. Either way, the first recommendation is to turn off logging of packets with both of the formerly reserved TCP bits set. This is standard ECN traffic, and the amount of it is increasing daily. The latest versions of the Linux kernel and Solaris both include ECN support. OpenBSD 3.2 includes the option to use ECN, but it is not quite functional in that release. (Cho) It is expected that the next release will include a working ECN implementation. Other operating systems may also include ECN support. With all of that going on, logging ECN packets is just creating noise for an analyst to fight through, and a lot of it at that.

The next issue is how to reduce the tremendous amount of false positives associated with use of the Snort http_decode preprocessor. The simple answer is to turn it off. This answer is naturally a bit too simplistic. The better idea would be to keep it turned on on the sensors right in front of the university's Web servers, and turn it off in other areas, for instance wherever people surf. This leaves open the possibility that students attacking outside Web servers will go undetected if their attacks would have been caught by the http_decode preprocessor, but this is far preferable to having so many false alarms that it's not worth the analyst's time to sort through them. Even this may not be enough, given the prevalence of Web attacks. Sorting through Web attacks is like reading through every NetBIOS name service scan against a large network: what's the point? Still, these attacks can be critical if they are successful (in contrast to NetBIOS name service scans), so they should not be ignored. The installation of reverse proxies in front of the Web servers that are capable of catching these attacks before they even get to the Web server would be a good investment of a few dollars. The IDS sensors should then be positioned between the reverse proxy and the Web servers. This solution has the added benefit of being able to cache answers from the Web servers, thus improving performance both for the surfer and the Web servers. One problem remains with this idea, and that is protecting the Web servers that appear to be installed on many end user systems, probably for the sake of ease of administration. Those Web servers need to be protected, too. This is better done

with some kind of local access control on the end user system, such as a personal firewall that blocks all access to port 80 except for access from certain administration addresses.

The astonishing thing about doing this analysis is that the log files indicate that no firewall is in place at the university. While the analyst is aware of the fact that a university setting prizes freedom, individuality, and standing on one's own two feet, it might still be prudent to put a basic firewall in place to block some of the more obvious attacks. A plain old stateful firewall will eliminate mapping attempts, malicious fragmentation, and other such minor attacks, even if all connections are let through the firewall. It's not much, but it could help. At least it would help keep down the number of alarms. While the firewall will almost certainly have to take a default open stance, some obviously harmful ports could be blocked, such as 65535 and 27374. Finally, outside access to certain machines that only have meaning to inside users (or administrators), like the kiosks and the printers could be entirely blocked without reducing functionality or flexibility.

If a firewall of any sort is put in place, it could also be a solution to the problem of too many false positives in the Web attacks category. Some advanced firewalls, like Check Point's FireWall-1, are capable of checking the URL being requested in an HTTP transaction and sending back an error message to the requester if the URL is forbidden (for example, a URL with default.ida in the path). The thing to be careful of in this constellation is that this function does not take so much CPU time on the firewall that overall throughput for all traffic travelling through the firewall is degraded.

Using a Web server on every end user system to manage a large network is not a bad idea, but those Web servers need to be protected better than they are. If that is indeed the intention of the discovered Web servers, they should definitely be running encrypted (HTTPS on TCP/443), and the best setup would be an internal PKI for the purpose of limiting access to the administrator interface. Every Web server that exists exclusively for administrative purposes should require a client certificate from the internal root CA, and perhaps an additional username/password. Only administrators would be given such a certificate. This effectively limits the attacks an attacker can even attempt, and would also drastically reduce the number of alarms. Encryption is obviously a good thing if this is an administrator interface, otherwise there are sure to be important data flowing in cleartext across the network. The machines that fell to Adore are proof that this approach would reap benefits. If HTTPS and X.509 are not an option, perhaps a separate administration LAN for certain components, like Web-enabled switches and routers, might be. This may already be the case, since no attacks against known switches and routers were observed.

There is one serious problem with the way the university's IDS is set up, and that is that it's not fast enough. There are many, many occurrences in the alerts files of one line being partially overwritten by the next line because the log buffer was full or the alarms came too fast. This reduces the integrity and usefulness of the logs. If the university intends to collect statistical data and to find the worst offenders, it would be much better to log to a binary format, like the Snort unified format or tcpdump/pcap format and to create the text

alerts, oos, and scan files later. Otherwise, the university needs to reduce the number of things it alerts on, and may need to rethink placement and numbers of IDS sensors/loggers.

The security of the public machines needs to be strengthened. Good configuration management, an initial install that makes attempted breakins very difficult, and an automated patching process after that is the best way to go about it.

User education must always be a priority on the university's list. As for the offenders, some of them may not realize that what they are doing is wrong, possibly illegal, and against school policy. What is not allowed should be clearly explained (repeatedly over time) with examples, and the penalties, both from the university and from the legal system, should be listed. This includes file sharing of copyrighted material. For the victims, they need to know what they can do to protect themselves. Sometimes it seems like security people are so wrapped up in their trade (and their hobby), that they forget that the things they consider basic in their field are not common knowledge. Most everyone knows nowadays that they should have virus protection, but most don't anyway. The minority knows that they need a firewall, and of those only a select few (the people who are professionals in the computer field anyway, and not even all of them) can do a decent job of configuring it. Certainly a seminar at the beginning of a student's career at the university should be mandatory, but repeated reminders in the form of newsletters, e-mail, or additional seminars (perhaps once a year) should also be considered.

While we're communicating with other people, some organizations that may have been used as stepping stones to attack the university's network, like SiteStream, Inc. and TerraLycos, should be contacted and informed that they may have a security problem on their hands. It's a good neighbor policy.

Back to the technical side, if the university is in the habit of employing watch lists, as it seems to be, and finds this mechanism valuable, it should consider adding the IP address ranges for the top attackers, abo.wandoo.fr and skynet.be, to the watch list. Conversely, the Chinese Academy of Sciences should probably be removed from the watch list, possibly after a "probation" period.

The attempts to access the VNC servers on the kiosks is disturbing, and for the short term the passwords should be checked to make sure they are of sufficient strength to withstand password guessing attacks. For the long term, a better technology needs to be employed to manage those machines. VNC is not encrypted, and only uses password authentication. Something like a session tunneled over SSH using public key cryptography/certificates for authentication would be a much better solution. As mentioned above in the paragraph about a firewall, these machines probably should not be accessible from the Internet, unless they need to be for administration purposes.

It was mentioned that the university does not participate in DShield. Participation is a highly recommended practice. It helps the entire Internet to identify the worst attackers

and take action against them. It is also an invaluable asset for providing corroborating evidence in the search for attackers on a smaller scale.

Finally, we turn our attention to the campus computers that are very likely already compromised. Based on the analysis, the following computers need at least to be examined, and they may all need to be formatted and reinstalled:

- MY.NET.150.5
- MY.NET.84.151
- MY.NET.88.193
- MY.NET.137.1
- MY.NET.137.35
- MY.NET.137.38
- MY.NET.137.17
- MY.NET.88.164

Beyond that, all passwords may need to be changed on the machines that experienced attempts to FTP the system password file to the attacker. Those machines are once again:

- MY.NET.5.92
- MY.NET.113.208
- MY.NET.130.187
- MY.NET.157.105

Conclusion

It is this author's hope that the analysis provided sheds much light on the current state of the university's network, and that the recommendations provided based on our improved understanding of the network will be taken seriously and considered for their immediate implementation. Universities are an especially difficult place to defend, because they have to balance security with the aspects of university life that are so prized, like freedom and experimentation. At the same time, one must also consider that university students are old and educated enough to know right from wrong, and to be held accountable for their actions.

Description of Analysis Process (a.k.a Many Thanks to the Following Tools)

During the course of this analysis, many tools were used and abused. The author wishes to thank (in no particular order): grep, egrep, cut, sort, uniq, sed, gnuplot, more, wc, vi, bash, tcsh, snort, ^C, StarOffice 6.0, and Jeffrey E.F. Friedl for writing "Mastering Regular Expressions" (regexes are a godsend). In addition, many audio CD's replaced caffeine in keeping the author awake and motivated. For that, the author would like to express gratitude to a great multitude of composers and performers, but especially to Kronos Quartet for their unceasing though unwitting support of research in intrusion detection.

After spending a little time getting a feel for what is in each of the log files, the analyst proceeded to identify relationships between campus machines by starting with the ECN entries in the out of spec files. These packets are normal traffic, and tell us what the function of a machine is, that is, what kind of traffic it should be seeing. Other out of spec entries were used to identify the webcams because the analyst noticed that the port 37 for time synchronization was involved.

The analyst then proceeded to generate a series of lists, almost all based on the scans files, many of which proved in the end to be of little use, and all simply one of or a combination of the source address, source port, destination address, and destination port. This approach was taken to start with because the analyst was still emotionally with the analysis process he went through for another large set of data he had recently sifted through, with which that technique worked relatively well. In the end, the best avenue of attack was to use the list of alerts, sorted by priority, as a entry place into understanding the data.

Having understood this, the analyst proceeded to filter through the alert logs with standard Unix tools to find all occurrences of the alert in question, identify large numbers of alerts that were probably false alarms (an obvious Web transfer or KaZaa traffic), filter those out, and look at the list again. This list was sometimes enough to look at in detail. If not, more automated techniques were used to identify the top talkers, source and destination addresses. Usually, the first few top talkers from both lists were identified, and an attempt was made to classify the traffic. This is especially true if the first entries in the list were head and shoulders above the other entries in terms of numbers of alerts. If a relationship between two machines in a series of alerts was not clear from the given alerts, the scans and out of spec logs, as well as other alerts with the same source or destination address, were pulled in to search for surrounding suspicious traffic. Close attention was paid to timing of the alerts and related scans and out of spec packets. At this point, it was often possible to make an educated guess as to the nature of the traffic. If port numbers were unknown, they were looked up using the IANA list of registered ports, Google, the Internet Storm Center, or DShield. If none of these sources turned up useful information about the port, and there did not seem to be any surrounding FTP traffic (which might indicate a passive FTP transfer using ephemeral ports on both sides), the traffic was deemed unknown and therefore suspect.

Analysis of the port scans was straightforward. Lists of top talkers and top source and destination ports were generated. First the destination port list was analyzed (with help from the source port list) and the most attacked ports were listed. This information is not normally useful for finding intrusions, but it gives the reader, the analyst, and the system administrator an idea of where most attacks are directed, and perhaps where efforts to secure the campus need to be directed. After that, lists of the top source and destination addresses based only on the port scans were generated essentially for the same purpose.

The last list generated was the list of hosts that generated the most alerts in the "high" danger category. This was done because these hosts are the most likely to wreak true havoc on the university's network.

After the analysis was done, the process of correlation with external sources began. Again, DShield was used to determine if the top ports and attackers at the university follow a general trend in the Internet. A handful of the most recent GCIA practicals were downloaded from GIAC and combed for corroborating evidence. These practicals were from Gary Smith, Bradley Urwiller, Brian Sheffler, Joe Ellis, Michael Holstein, Rick Yuen, Steven Drew, and Tod Beardsley. This did not bring much, as the newest available practicals are from six months ago. The university may be in a bad situation security-wise, but the environment seems to be very dynamic. Yesterday's hackers are not today's, who are not tomorrow's. This completed the analysis process. All that remained was to write the recommendations and management summary.

Bibliography for Part III

Beardsley, Tod. "Intrusion Detection and Analysis: Theory, Techniques, and Tools". Global Information Assurance Certification. 8 May 2002. 9 February 2003. <http://www.giac.org/practical/Tod_Beardsley_GCIA.doc>.

Brentano, Joshua. "Online Games and Their Port Assignments". 27 April 2001. TechTV (www.techtv.com). 8 February 2003. <http://www.techtv.com/screensavers/answerstips/story/0,24330,3324625,00.html>.

"CAMPUS MAP". The University of Maryland, Baltimore County. 8 February 2003. <http://www.umbc.edu/aboutumbc/campusmap/BC2.html>.

Cho, Kenjiro. "Re: ecn and 3.2, no ECONNRESET?". 29 January 2003. misc@openbsd.org. SigmaSoft, Incorporated. 9 February 2003. <http://www.sigmasoft.com/~openbsd/archive/openbsd-misc/200301/msg01746.html>.

Drew, Steven. "Intrusion Detection in Depth". Global Information Assurance Certification. 4 June 2002. 8 February 2003. <http://www.giac.org/practical/Steven_Drew_GCIA.doc>.

"DShield - Port Report". DShield. Euclidian Consulting. 9 February 2003. <http://www.dshield.org/port_report.php>.

Ellis, Joe. "GIAC Practical Assignment v3.0: Intrusion Detection in Depth". Global Information Assurance Certification. 14 May 2002. 8 February 2003. <http://www.giac.org/practical/Joe_Ellis_GCIA.doc>.

Holstein, Michael. "SANS GCIA Practical Assignment". Global Information Assurance Certification. 5 June 2002. 8 February 2003. <http://www.giac.org/practical/Michael_Holstein_GCIA.doc>.

"Internet Storm Center: Port 6257". Internet Storm Center. 9 February 2003. <http://isc.incidents.org/port_details.html?port=6257&recax=1&tarax=2&srcax=2&percent=N>.

"NIMDA Worm/Virus Report -- Final". The SANS Institute. 3 October 2001. www.incidents.org. 9 February 2003. <http://www.incidents.org/react/nimda.pdf>.

Office of Information Technology. "Acceptable Use Policy". The University of Maryland, Baltimore County. 6 September 1996. 8 February 2003. <http://www.umbc.edu/oit/sans/security/policy/2-UMBC/IT-01-final.html>.

"Ports and Known/Suspected Service: Registered Ports (1024-49151)". 18 April 2001. The University of Maryland, Baltimore County. 8 February 2003. <http://userpages.umbc.edu/~robin/Security/portlist-1024-49151.html>.

Rautiainen, Sami. "F-Secure Computer Virus Information Pages: Adore". F-Secure. April 2001. 8 February 2003. <http://www.f-secure.com/v-descs/adore.shtml>.

Rothschild, Matthew. "Nimda -- A Step Into Complexity". The SANS Institute. 27 November 2001. 9 February 2003. <http://www.sans.org/rr/malicious/complexity.php>.

"SBL5817". The Spamhaus Project. 16 December 2002. 8 February 2003. <http://www.spamhaus.org/sbl/sbl.lasso?query=SBL5817>.

Sheffler, Brian K. "Intrusion Detection in Depth: GIAC Certified Intrusion Analyst (GCIA) Practical Assignment". Global Information Assurance Certification. 26 March 2002. 9 February 2003. <http://www.giac.org/practical/Brian_Sheffler_GCIA.zip>.

Smith, Gary. "Analyze This: Intrusion Detection in Depth". Global Information Assurance Certification. 10 June 2002. 9 February 2003. <http://www.giac.org/practical/Gary_Smith_GCIA.zip>.

TonikGin (pseudo.). "XDCC -- An .EDU Admin's Nightmare". 11 September 2002. 8 February 2003. <http://www.russonline.net/tonikgin/EduHacking.html>.

Tracker (pseudo.). "PORT NUMBER AND SERVICES". 21 December 2002. comp.security.firewalls. Der Keiler.de (www.der-keiler.de). 8 February 2003. <http://www.der-keiler.de/Newsgroups/comp.security.firewalls/2002-12/2017.html>.

Urwiller, Bradley D. "SANS GIAC Globally Certified Intrusion Analyst (GCIA)". Global Information Assurance Certification. 23 April 2002. 8 February 2003. <http://www.giac.org/practical/Bradley_Urwiller_GCIA.pdf>.

watford(hi2u)@uiuc.edu. "Port reference". sloppycode.net. 13 August 2002. 9 February 2003. <http://www.sloppycode.net/ports/>.

Yuen, Rick Winkey. "GIAC Intrusion Detection, Practical Assignment". Global Information Assurance Certification. 11 October 2001. 8 February 2003. <http://www.giac.org/practical/Rick_Yuen_GCIA.doc>.