Bro Covert Channel Detection (BroCCaDe) Framework: Scope and Background

Hendra Gunadi (Hendra.Gunadi@murdoch.edu.au),
Sebastian Zander (s.zander@murdoch.edu.au)

17 November 2017

Abstract

Our goal is to extend Bro with mechanisms to detect covert channels. In this report, we investigate and compare different ways to extend Bro and propose the design of our Bro extension. Bro is very flexible and there are multiple ways to extend it. We first review the advantages and disadvantage of each approach. We find that for our purposes extending Bro with a plugin is the most feasible approach. Then we describe the pattern of covert channels we are interested in, and present the analysis metrics our extension needs to support to detect these patterns. Finally, we propose the design of our Bro extension.

1 Introduction

In technical report [12], we determined that the most suitable basis for our covert channel detection IDS is the Bro IDS. In this report, we first define the scope of the project, such as the types of covert channels we consider, and terminology used throughout the report. Next, we explore different possibilities to extend Bro [5]. Bro is very flexible and there are multiple ways to extend it, for example, by writing a Bro script, by writing a C program (through Broccoli [2]), or by extending Bro with a plugin (written in C++). We discuss the advantages/disadvantages of each approach and find that implementing a plugin to is the most flexible approach and best suited to our needs. Finally, we discuss in more detail the covert channels we consider and the analysis metrics we will implement to detects these types of covert channels.

Section 2 defines the scope of our project. Section 3 defines the terminology we refer to in the rest of the report. Section 4 discusses the available extension mechanisms that Bro offers. Section 5 and Section 6 provide the background theory used in our framework, detailing the covert channel patterns that we are going to focus on and the analysis metrics we are going to use to detect covert channels. Section 7 concludes the report.

2 Project Scope

Since the field of covert channels is a broad area, here we specify the exact scope of our project:

- Our project is about detecting covert channels, and we are not really concerned with how to deal with detected covert channels.
- We decided implementing the covert channel detection in Bro, so we are limited by Bro's capabilities. Although in general Bro is very powerful, it has some limitations. For example, with Bro we cannot easily handle protocols below the network layer, so we focus on covert channels on the network-layer and above. These are more critical anyway as they have further reach than link-layer channels.
- There are already a lot of protocol parsers available in Bro, but not every protocol has a parser for it. If one wants to extend the framework to deal with a covert channel in a protocol not already supported by Bro, a new parser for this protocol has to be written.

- We will focus on covert channels in network protocols based on the surveys in [18,20]. We are not aiming to detect covert channels in single host environments (i.e., between different processes on the same host), channels created by steganographic techniques for hiding information in content (e.g. images, audio, video), or subliminal channels in cryptosystems.
- While we aim to implement the framework such that it can handle all sorts of different covert channel patterns [18], for the initial prototype implementation we are not aiming to cover all possible patterns. We choose some patterns for our proof of concept implementation, and the rest of the patterns are left for future development. We decided that we will initially focus on the following patterns: size modulation pattern, value modulation pattern, reserved/unused pattern, inter-arrival time pattern, and rate pattern. These five patterns cover a lot of existing covert channels [18] and they are a good representative for both timing covert channels and storage covert channels.
- The core of our detection lies in the ability to analyse the traffic, but we shall limit ourselves to several detection metrics that have been used previously in the literature (e.g. [11, 18]) to detect covert channels: Kolmogorov-Smirnov Test, Entropy analysis, Corrected Conditional Entropy (CCE), Multi Modal analysis, Sum of Autocorrelation, and Regularity analysis.

3 Terminology

Here we define several terms that we use throughout the rest of the report as well as in future reports.

A *Bidirectional Flow* is determined by transport protocol related criteria and a 5-tuple ID of source and destination IP address, protocol number, and source and destination port. All of the traffic that belongs to the same 5-tuple ID (or its reverse) belongs to the same flow if it also fulfils the criteria. For TCP, the criteria involve the state corresponding to the type of packets sent, e.g. setting up a flow with a SYN packet, tearing it down with a FIN packet, etc. For UDP, a timeout is used to determine whether a flow has ended, i.e. if there are no packets in either direction before the timeout, then a new packet with the same 5-tuple (or its reverse) will start a new flow. We basically use Bro's definition of a flow. As a side note, we refer to 5-tuple ID of source and destination address, protocol, and source and destination port, but Bro implements it as a 4-tuple. Bro collapse the protocol and the ports together, e.g. outgoing DNS packet which uses UDP protocol will have its destination port value of "53/UDP" as opposed to "53".

A *Unidirectional Flow* has the same criteria as a bidirectional flow, except that only packets with the same 5-tuple ID are considered as part of the flow (does not consider the reverse of the 5-tuple ID).

A *Feature* is useful information that can be extracted from the packet data parsed by a protocol parser. A feature will take the form of a *vector* of numerical values (note that characters can also be expressed as numerical values, for example ASCII values). The associated computation to produce the feature depends on the type of feature. In its simplest form, a feature can be the value of some packet header field.

A *Detection Metric* is calculated from a feature based on some analysis method. A really simple analysis metric to detect the use of unused header bits as covert channel is to sum the values of the feature vector. If the sum is non-zero, then we know that the flow is suspicious. An example for a more complex metric is calculating the entropy of the feature value and use it to determine whether a flow contains a covert channel or not.

Threshold refers to a limit (either lower bound or upper bound) of the value of a detection metric. If the metric is below or above the threshold, then we can suspect that a flow may contain a covert channel.

4 Extension Mechanisms in Bro

There are three different "standard" ways to extend the Bro IDS:

1. Write a custom Bro script [7] and do the analysis directly in the script.
2. Write a C/C++ program to communicate with a Bro instance using Broccoli [2].

3. Extend Bro's functionality by writing a plugin [3, 6].

Bro scripts are event driven, that is, the programming paradigm revolves around *events* and *event handlers*. An event in Bro is represented as an object which is raised whenever an event of interest happens. Some examples of events in Bro are *bro_init(), tcp_packet(),* and *ssl_client_hello(),* which are raised when Bro starts, a TCP packet arrives or an SSL hello message from a client is observed. However, an event is only raised when there is an existing event handler for it. We can write a script in Bro to register our interests in a particular event, and do a computation based on the information passed in the event.

Writing BroCCaDe as Bro script(s) is quite appealing. Bro script is Turing complete, meaning that it can be used to code anything that we want (as long as it is programmable). Bro script is easy to implement and there already exists an extensive list of events that Bro script can handle. However, this approach has some drawbacks:

- Bro scripts execute much slower than native C/C++ code meaning there will be a significant performance overhead compared to native code.
- There is not much library code we could use. For example, we would have to reimplement machine learning techniques which are readily available in other languages.
- There is a limitation in that we can only make use of the information that is provided in Bro events.

Writing a C/C++ program to communicate with a Bro instance has the advantage of using much faster native code in a separate standalone application that communicates with Bro via a well-defined interface in the form of event handling and event queuing. The problem with this approach is that Broccoli is deprecated and will no longer be supported in future Bro versions [1].

This plugin mechanism still benefits from the events that Bro generates. The plugin interacts with the Bro script to provide "native" functionality, hence it still can handle Bro generated events (or events generated from other plugins). While it is more complicated than writing a Bro script, the plugin mechanism has several benefits. Firstly, there are a lot of analysis libraries written in C/C++ that we can use, whereas we would have to develop these from scratch if we would use a pure Bro script. Secondly, with C/C++ the performance of BroCCaDe will be much better than with pure script programming. Lastly, we can write a plugin which extracts information that Bro does not give us, e.g. the TCP header's URG pointer field.

The plugin approach provides the best performance, will be supported in future Bro versions and is also the most flexible approach. Hence, we decided to implement BroCCaDe as a plugin.

5 Covert Channel Patterns

In this section we discuss the covert channel patterns we considered in Section 2.

5.1 Size Modulation Pattern

A covert channel of this pattern hides covert information by manipulating the size of a protocol data unit (PDU) and adjusting size-related header fields within a PDU as necessary. An example to this pattern is a simple packet length covert channel where the sender and the receiver agree beforehand on the symbols used. In the binary case, a packet with a particular length signifies a logical zero (0-bit) and another length signifies a logical one (1-bit). To be more concrete, suppose that a packet of length *70* represents a 0-bit and a packet of length *100* represents a 1-bit. In this scenario, if the sender wants to convey *11010,* the sender will send packets with lengths of *100, 100, 70, 100* and *70*. We can extend the channel to multiple symbols, so that each length represents more than one bit of information.

Another example that falls into this pattern is a work by Ji et al. called Normal-traffic Network Covert Channel (NTNCC) [14]. Covert sender and receiver agreed beforehand on the number of so-called *Reference* packets and the number of buckets *b*. During initialisation, the covert sender sends the agreed number of *Reference* packets to the covert receiver. Both the covert sender and the covert receiver sort the *Reference*

packets based on the packet lengths and distribute them into b roughly equal-sized buckets of packets, effectively creating a "dictionary" that maps buckets to covert bit strings. To send a covert bit string, the covert sender picks a possible packet length from the corresponding bucket and sends a packet with that length to the covert receiver. The receiver then decodes the covert data by inspecting which bucket contains the received packet length.

5.2 Value Modulation Pattern

This pattern consists of covert channels that hide covert information by setting the value of a header element to correspond to a particular symbol. For example, the covert sender embeds covert information in the IP TTL field [19]. In this channel, the encoding depends on two values *high-TTL* and *low-TTL*. High-TTL encodes a logical 0 while low-TTL encodes a logical 1. The actual encoding can vary as described in [19]. For example, the value of high-TTL can be the default initial TTL and the value of low-TTL can be high-TTL minus one.

5.3 Reserved / Unused Pattern

This particular pattern hides information in reserved / unused fields in a PDU because the value of these field is rarely interpreted by network stack implementations. This type of channel is the easiest to detect. Usually, not every bit of information in the packet header is used / interpreted by the network stack implementation and in the case it is not used, it should be set to a default value. For example, if the urgent flag is not set, then the urgent pointer is not used and can take any arbitrary value. In this scenario, we only need to compare the value of the unused headers against their default values.

We choose Ping Tunnel [4] as a representative for this pattern because of its simplicity. Ping tunnel encodes its secret data into the ICMP echo payload, and the packets containing covert data are identified by setting the first four bytes of the payload to the magic number ($0xD5200880$). The Ping Tunnel [4] prototype was built to provide a tunnel allowing the tunnelling of TCP packets over the covert channel. At the moment Ping Tunnel only supports TCP.

To set up this covert channel, a pair of hosts is required – one being the covert sender and the other being the covert receiver. The sender sends ICMP echo requests to the receiver with the first four payload bytes set to the magic number and the rest of the payload being the (full / partial) content of TCP packets. The covert receiver checks whether incoming ICMP packets contain the magic number. Based on the packets with the magic number, the receiver will reconstruct the TCP packets and send them to the final destination (real receiver). The covert receiver forwards return TCP traffic from the real receiver to the covert sender via the same mechanism. Thus for this channel the covert receiver is actually a proxy.

5.4 Inter-arrival Time Pattern

There are different ways to encode covert channels using inter-packet gaps, but here we consider two different techniques. The first one is a simple binary symbol inter-packet gap covert channel as proposed in [8]. This covert channel encodes its hidden messages through inter-packet delay. A short gap between two packets is interpreted as logical zero and a long gap between two packets is interpreted as logical one.

The second channel is the channel proposed by Shah et al. [16], Jitterbug, where covert information is encoded in the modulus of inter-packet gaps. Jitterbug modulates the timing between between keystrokes delivered from the keyboard to the host. If these keystrokes are part of an SSH session, this also modulates the inter-packet gaps of the packets carrying the SSH keystrokes. Then, the covert receiver can recover the covert information from observed inter-arrival times between packets. For a pre-defined w, the covert sender must delay each packet such that $\delta_i \bmod w = \begin{cases} 0 & \text{if } b_i = 0; \\ \lfloor w/2 \rfloor & \text{if } b_i = 1; \end{cases}$, where δ_i is the time gap between the last input and the current input and b_i is the current covert bit to convey. The covert receiver simply needs to observe the packets from the covert sender and calculate the values of the inter-arrival times modulus w to decode the covert message.

5.5 Rate Pattern

Information can be hidden in the throughput / rate of the packets sent. Cabuk et al. proposed an on/off packet rate timing channel [10]. For this covert channel, the covert sender and receiver both agree on a timing interval length and the starting protocol, an event which signifies the start of the transmission. The sender will either send or stay idle during a time interval depending on whether the covert is a logical one or logical zero. Sending at least one packet within the time interval indicates a logical one while sending no packets indicates a logical zero. The receiver will monitor the traffic in each time interval and check whether there is a packet from the covert sender or not and decode the covert bits accordingly.

6 Analysis Metrics

In this section we explain the analysis metrics that we will use in our infrastructure.

6.1 Kolmogorov-Smirnov Test (KS Test)

In general, the KS test is used to measure the distance between two distributions, usually between the data set of interest and either a known distribution or a data set to compare against. The rationale of using this test as a way to detect covert channels is that the embedding of a covert channel may change the carrier channel's characteristics, and the resulting distribution of a feature variable deviates from the distribution of a legitimate channel. Since we do not know the distribution of the channel a priori, we choose to compare the data set of interest against a data set captured without covert channels.

The calculation of the KS test itself is captured with the following formula

$$D = \max_{-\infty<x<\infty} |S_{N_1}(x) - S_{N_2}(x)|$$

where $S_N(x)$ is the cumulative distribution estimation which indicates the fraction of data points that is smaller than x. In practice, we do not iterate over all possible values. Instead, the cumulative distribution estimation is obtained by sorting the data and we iterate over all the indices of the data in both of the data sets. The significance of the KS test is calculated as

$$probability(D > observed) = Q_{KS}([\sqrt{N_e} + 0.12 + 0.11/\sqrt{N_e}] \cdot D)$$

$$Q_{KS(\lambda)} = 2\sum_{j=1}^{\infty}(-1)^{j-1}e^{-2j^2\lambda^2},\ Q_{KS}(0) = 1,\ Q_{KS}(\infty) = 0\ ,$$

where $N_e = \frac{N_1 N_2}{N_1+N_2}$ is the effective number of data points, N_1 is the number of data points in the first data set, and N_2 is the number of data points in the second data set.

6.2 Entropy

Entropy is a measure of the uncertainty of a data distribution, i.e., it is a measure of the disorder/randomness of the data. Entropy analysis allows to test whether different distributions have a similar shape or not. We can compare the entropy of a tested dataset with the entropy calculated previously for examples of legitimate traffic flows and covert channels. If the entropy of the test data is similar to the entropy values observed for one of the two classes, we can classify the test instance as belonging to that class. The entropy itself is calculated as follows [17]:

$$H(X) = -\sum_{x\in X} P(x)\log P(x))\ .$$

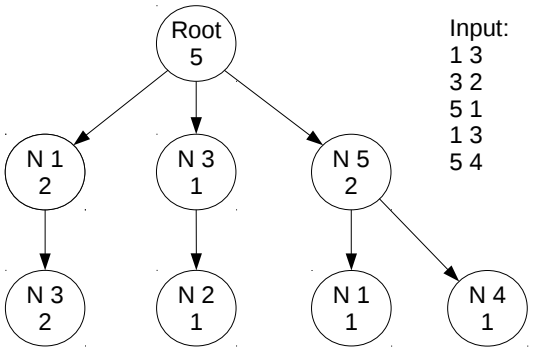

Figure 1: Tree Data Structure

6.3 Corrected Conditional Entropy

Gianvecchio and Wang proposed to use Corrected Conditional Entropy (CCE) to detect packet timing covert channels [11]. CCE is used to estimate the entropy rate of a finite dataset [15]. CCE is a test of regularity because the entropy rate is the uncertainty of predicting the next input given the input pattern so far. When the uncertainty is high, then the data set is irregular and less likely to be a covert channel. When the uncertainty is low then the data set has some regularity in it, which could mean a covert channel is present.

The CCE itself is defined as

$$CCE(X_m|X_1,\cdots,X_{m-1}) = CE(X_m|X_1,\cdots,X_{m-1}) + perc(X_m)\cdot H(X_1)$$

where $CE(X_m|X_1,\cdots,X_{m-1})$ is the conditional entropy of pattern X_m given $X_1,\cdots,X_{m-1}$, patterns of length $1,\cdots,m-1$, $perc(X_m)$ is the percentage of unique patterns of length m, and $H(X_1)$ is the entropy of pattern of length 1. Conditional entropy itself is calculated as

$$CE(X_m|X_1,\cdots,X_{m-1}) = H(X_1,\cdots,X_m) - H(X_1,\cdots,X_{m-1}),$$

$$H(X_1,\cdots,X_m) = -\sum_{X_1,\cdots,X_m} P(x_1,\cdots,x_m)\log(P(x_1,\cdots,x_m))$$

where $P(x_1,\cdots,x_m)$ is the probability (estimate) of a specific pattern of length m. With the CCE calculation taking into account the percentage of unique patterns, it will cancel out the effect of longer patterns where it will be more likely to observe unique patterns (i.e., the entropy is zero).

Gianvecchio and Wang implement the analysis using equiprobable bins[1] and storing the patterns as a tree with the depth equal to the pattern length. The number of children of each node is equal to the number of bins. The tree itself will take a pattern of length L, where L is the pattern length for CCE (or the depth of the tree). Each incoming pattern will increase the counters of all nodes in the tree that make up that pattern (see Figure 1).

We calculate the CCE for each level of the tree in a breadth-first traversal, and then find the minimum value which is the estimate of the entropy rate.

6.4 Multi Modal Analysis

Multi modal analysis is used to estimate the number of modes in the data distribution. It is also based on using a histogram of the data. Since for meaningful covert information there must be some representation of covert

[1] Equiprobable binning is a strategy to set the boundary of the bins according to the distribution of the data, i.e. each bin (roughly) has the same number of data points.

bits as channel symbols, the number of modes can indicate these meaningful symbols. The estimate is done by calculating [13]

$$M(X) = \frac{\sum n_i^2}{\max(n_i^2)} ,$$

where n_i is the frequency of bin i. The n_i are squared to emphasise dominant symbols and fade less frequent symbols.

6.5 Autocorrelation Analysis

Following [13], we also use the sum of autocorrelation coefficients as an analysis metric. The autocorrelation coefficient R_τ indicates whether there is a correlation between a series of data and a delayed copy of the data, delayed by the lag τ. The sum of the autocorrelation coefficients ρ is the sum of autocorrelation for various lags

$$\rho_A = \frac{1}{N} \sum_{\tau}^{A} |R_\tau|$$

$$R_\tau = \frac{E[(Y_t - \mu)(Y_{t+\tau} - \mu)]}{\sigma^2} ,$$

where A is the set of autocorrelation lags, N is the number of elements in A, τ is the lag, $Y_1, \cdots, Y_N$ are the various data points in the data set, μ is the mean, $\sigma^2$is the variance, and E is the expected value. The idea behind the autocorrelation analysis is that with legitimate traffic some features exhibit a particular self-correlation pattern. Some covert channels destroy the autocorrelation of the original data series [21], so a lack of autocorrelation may signify that a covert channel is used.

6.6 Regularity Analysis

Cabuk et al. proposed a regularity metric [9] to detect packet timing covert channels, defined as

$$regularity = STDEV(\frac{|\sigma_i - \sigma_j|}{\sigma_i}, i < j, \forall \text{i,j}) .$$

The metric is measuring the standard deviation of the standard deviations of different parts of the data set, which in essence is showing whether the variance in the data set is constant over time or not. The reasoning is that if a covert channel exists, there must be some sort of regularity or pattern. This implies that a high regularity means that the data set is highly irregular, thus it is less likely to be a covert channel. Low regularity means that the data set has some regularity, thus there might be a covert channel embedded.

7 Conclusions

Based on the advantages and disadvantages of methods to extend Bro, we decided to use the plugin mechanism for BroCCaDe. In this report we have also defined the scope of our project, in particular the patterns of covert channels that we are interested in, i.e. size modulation, value modulation, reserved / unused, inter-arrival time, and rate patterns. We have also described the analysis metrics that we are going to use to detect covert channels, i.e., Kolmogorov-Smirnov Test, Entropy, CCE, Multi Modality, Autocorrelation, and Regularity analysis.

Acknowledgements

This work was supported by a grant from the Comcast Innovation Fund.

References

[1] [bro] broccoli code not working : Not receiving any events. http://mailman.icsi.berkeley.edu/pipermail/bro/2016-June/010022.html, Accessed: 27 March 2017.

[2] Broccoli: The bro client communications library. https://www.bro.org/sphinx/components/broccoli/broccoli-manual.html, Accessed: 27 March 2017.

[3] Extending Bro with Builtin Functions (BiF). https://www.bro.org/development/howtos/bif-doc/, Accessed: 27 March 2017.

[4] Ping tunnel - for those times when everything else is blocked. http://www.cs.uit.no/~daniels/PingTunnel/, Accessed: 6 April 2017.

[5] The Bro Network Security Monitor. https://www.bro.org/, Accessed: 8 March 2017.

[6] Writing Bro Plugins. https://www.bro.org/sphinx-git/devel/plugins.html, Accessed: 27 March 2017.

[7] Writing Bro Scripts. https://www.bro.org/sphinx/scripting/index.html, Accessed: 27 March 2017.

[8] V. Berk, A. Giani, G. Cybenko, and N. Hanover. Detection of covert channel encoding in network packet delays. Technical Report TR536, University of Dartmouth, 2005.

[9] S. Cabuk, C. E. Brodley, and C. Shields. Ip covert timing channels: design and detection. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 178–187. ACM, 2004.

[10] S. Cabuk, C. E. Brodley, and C. Shields. IP covert channel detection. *ACM Transactions on Information and System Security (TISSEC)*, 12(4):22, 2009.

[11] S. Gianvecchio and H. Wang. An entropy-based approach to detecting covert timing channels. *IEEE Transactions on Dependable and Secure Computing*, 8(6):785–797, 2011.

[12] H. Gunadi and S. Zander. Comparison of IDS Suitability for Covert Channels Detection. Technical Report 20170818A, Murdoch University, 2017.

[13] F. Iglesias, R. Annessi, and T. Zseby. DAT detectors: uncovering TCP/IP covert channels by descriptive analytics. *Security and Communication Networks*, 9(15):3011–3029, 2016.

[14] L. Ji, H. Liang, Y. Song, and X. Niu. A normal-traffic network covert channel. In *International Conference on Computational Intelligence and Security (CIS)*, pages 499–503, 2009.

[15] A. Porta, G. Baselli, D. Liberati, N. Montano, C. Cogliati, T. Gnecchi-Ruscone, A. Malliani, and S. Cerutti. Measuring regularity by means of a corrected conditional entropy in sympathetic outflow. *Biological cybernetics*, 78(1):71–78, 1998.

[16] G. Shah, A. Molina, and M. Blaze. Keyboards and covert channels. In *USENIX Security Symposium*, volume 15, 2006.

[17] C. E. Shannon and W. Weaver. *The mathematical theory of communication*. University of Illinois press, 1998.

[18] S. Wendzel, S. Zander, B. Fechner, and C. Herdin. Pattern-based survey and categorization of network covert channel techniques. *ACM Computing Surveys (CSUR)*, 47(3):50, 2015.

[19] S. Zander, G. Armitage, and P. Branch. Covert channels in the IP time to live field. In *Australian Telecommunication Networks and Application Conference (ATNAC)*, 2006.

[20] S. Zander, G. Armitage, and P. Branch. A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys & Tutorials*, 9(3):44–57, 2007.

[21] S. Zander, G. Armitage, and P. Branch. Stealthier inter-packet timing covert channels. In *IFIP Networking*, pages 458–470. Springer, 2011.