Bro intrusion detection system - Principles of operation and internal structure

Slobodan Petrović
NISlab, Gjøvik University College

Overview of IDS/IPS

- Intrusion
  - A set of actions aimed at compromising the security goals (confidentiality, integrity, availability of a computing/networking resource)
- Intrusion detection
  - The process of identifying and responding to intrusion activities
- Intrusion prevention
  - The process of both detecting intrusion activities and managing responsive actions throughout the network

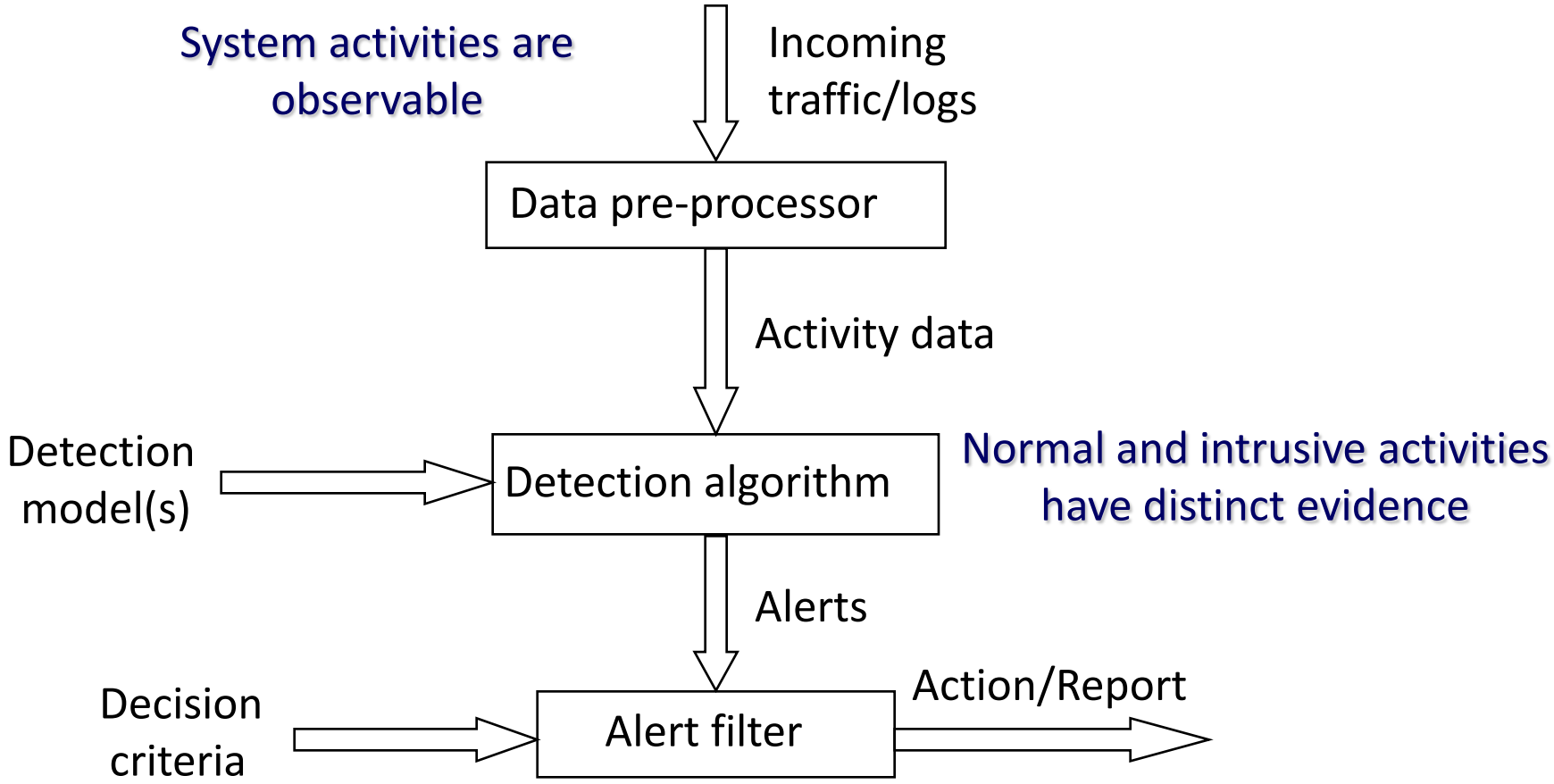
Overview of IDS/IPS

- Intrusion detection system (IDS)
  - A system that performs automatically the process of intrusion detection
- Intrusion prevention system (IPS)
  - A system that has an ambition to both detect intrusions and manage responsive actions
  - Technically, an IPS contains an IDS and combines it with preventive measures (firewall, antivirus, vulnerability assessment) that are often implemented in hardware

Overview of IDS/IPS

- Basic assumptions regarding operation of IDS
  - System activities are observable
  - Normal and intrusive activities have distinct evidence – the goal of an IDS/IPS is to detect the difference

Components of an IDS/IPS



Components of an IDS/IPS

- Data pre-processor
  - Collects and formats the data to be analyzed by the detection algorithm
- Detection algorithm
  - Based on the detection model, detects the difference between ”normal” and intrusive traffic

Components of an IDS/IPS

- Alert filter
  - Based on the decision criteria and the detected intrusive activities, estimates their severity and alerts the operator/manages responsive activities (usually blocking)

IDS/IPS classification

- By scope of protection (or by location)
  - Host-based IDS
  - Network-based IDS
  - Application-based IDS
  - Target-based IDS
- By detection model
  - Misuse detection
  - Anomaly detection

IDS classification

- Host-based
  - Collect data from sources internal to a computer, usually at the operating system level (various logs etc.)
  - Monitor user activities
  - Monitor execution of system programs

IDS classification

- Network-based
  - Collect network packets
    - This is usually done by using network devices that are set to the *promiscuous mode*
      - A network device operating in the promiscuous mode captures all network traffic accessible to it, not just that addressed to it
  - Have sensors deployed at strategic locations
  - Inspect network traffic
  - Monitor user activities on the network

IDS classification

- Application-based
  - Collect data from running applications
  - The data sources include application event logs and other data collections internal to the application

IDS classification

- Target-based (integrity verification)
  - Generate their own data (by adding code to the executable, for example)
  - Use checksums or cryptographic hash functions to detect alterations to system objects and then compare these alterations to a policy
  - Trace calls to other programs from within the monitored application

IDS classification

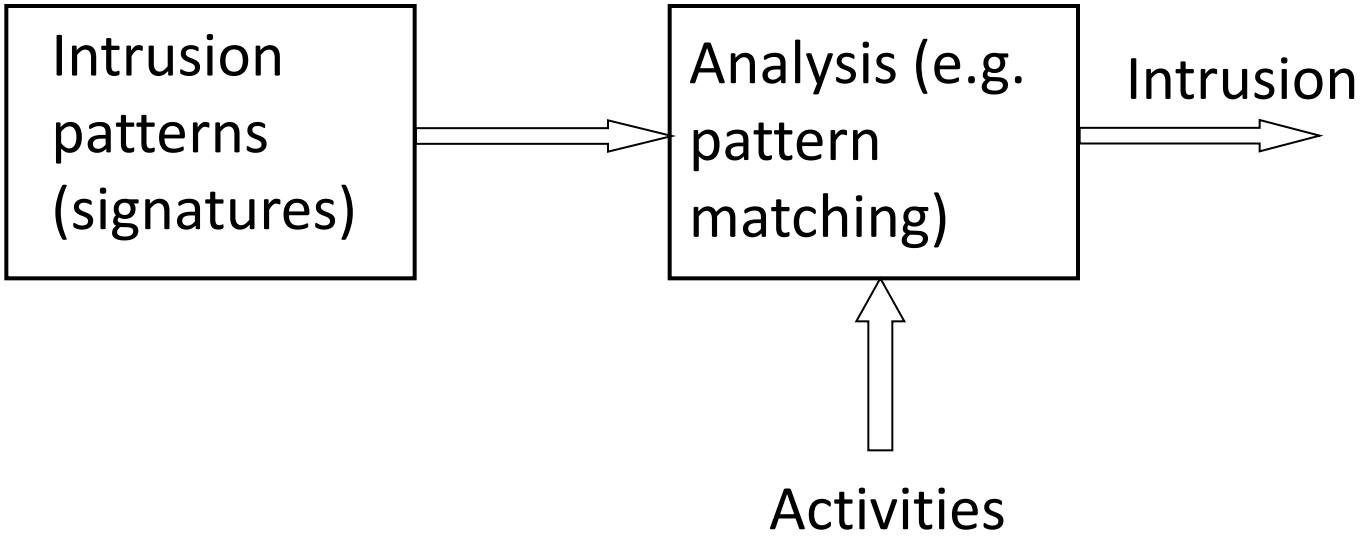
- Misuse detection (1)
  - Involves gathering information about indicators of intrusion in a database and then determining whether such indicators can be found in incoming data

IDS classification

- Misuse detection (2)
  - To perform misuse detection, the following is needed
    - A good understanding of what constitutes a misuse behaviour (intrusion patterns, or signatures)
    - A reliable record of user activity
    - A reliable technique for analyzing that record of activity (very often – pattern matching)

IDS classification

- Misuse detection (3)



Signature example: **if** src_ip = dst_ip **then** “land attack”

IDS classification

- Misuse detection (4)
  - It is best suited for reliably detecting known misuse patterns (by means of signatures)
  - It is not possible to detect previously unknown attacks - a single bit of difference in misuse patterns may be enough for an IDS to miss the attack
  - However, it is possible to use the existing knowledge (for instance, of outcomes of attacks) to recognize new forms of old attacks

IDS classification

- Misuse detection (5)
  - Misuse detection systems sometimes generate alerts even if the activities are normal (normal activities often closely resemble the suspicious ones)
  - Consequently, IDS that use signature detection are likely to generate a lot of *false positives*
  - Careful adjustment of the IDS parameters is needed to reduce the number of false positives

IDS classification

- Misuse detection (6)
  - New attacks require new signatures
  - The increasing number of vulnerabilities causes that signature databases grow over time
  - Every packet must be compared to each signature for the IDS to detect intrusions
  - This may become computationally expensive as the bandwidth increases

IDS classification

- Misuse detection (7)
  - When the bandwidth overwhelms the capabilities of the IDS, it causes the IDS to miss or drop packets
  - In such a situation, *false negatives* are possible

IDS classification

- Anomaly detection (1)
  - Involves a process of establishing profiles of normal user/network behaviour, comparing actual behaviour to those profiles, and alerting if deviations from the normal behaviour are detected
  - The basis of anomaly detection is the assertion that abnormal behaviour patterns indicate intrusion

IDS classification

- Anomaly detection (2)
  - Profiles are defined as sets of metrics - measures of particular aspects of user/network behaviour
  - Each metric is associated with a threshold or a range of values

IDS classification

- Anomaly detection (3)
  - Anomaly detection depends on an assumption that users/networks exhibit predictable, consistent patterns of system usage
  - The approach also accommodates adaptations to changes in user/network behaviour over time

IDS classification

- Anomaly detection (4)
  - The completeness of anomaly detection depends on the selected set of metrics – it should be rich enough to express as much of anomalous behaviour as possible
  - Capable of detecting new attacks

IDS classification

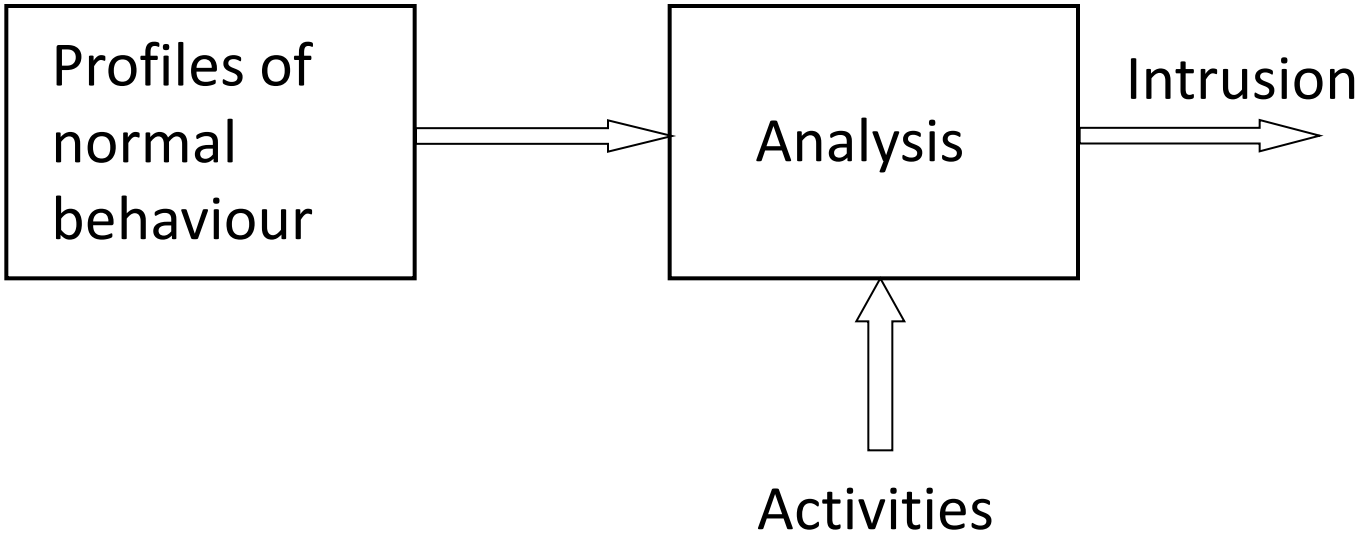
- Anomaly detection (5)
  - An attacker can replicate a misuse detection system and check which signatures it detects
  - Then the attacker can use the attack not detectable by the IDS in question
  - This is not possible to do with an anomaly detection system

IDS classification

- Anomaly detection (6)
  - However, it is not always the case that abnormal behaviour patterns indicate an intrusion – sometimes, rare traffic sequences represent normal behaviour
  - This is a major problem in anomaly detection – *false positives*
  - If anomaly detection IDS thresholds are set too high, we may miss the attacks and have *false negatives*

IDS classification

- Anomaly detection (7)



IDS classification

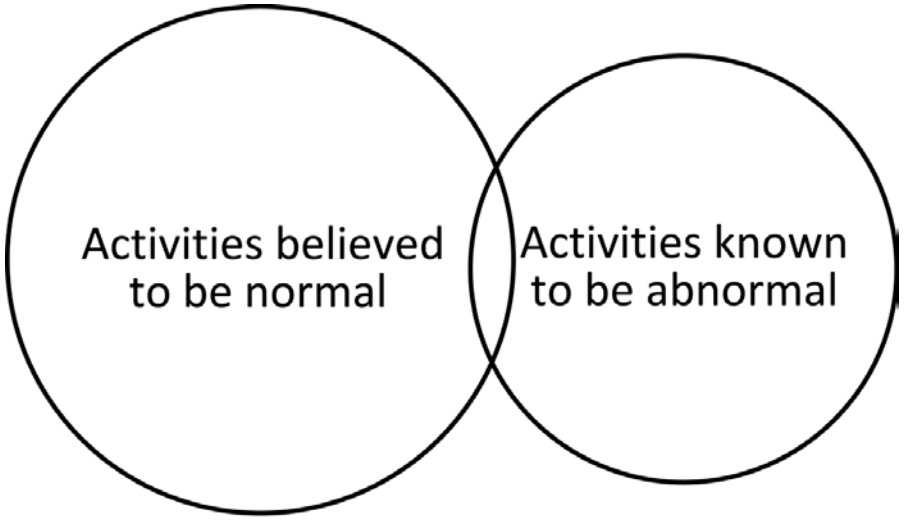
- Anomaly detection (8)
  - In misuse detection, the analysis engine alerts if the analyzed activity *matches* an entry in the signature database
  - In anomaly detection, the analysis engine alerts if the analyzed activity *does not match* any of the established profiles of normal behaviour
  - In both cases, search through the whole database is needed for each analyzed activity (e.g. for each packet)

IDS classification

- Anomaly detection (9)
  - Methods of anomaly detection
    - Statistical methods
    - Artificial intelligence (cognitive science,…)
    - Data mining
    - Mathematical abstractions of biological systems (neural nets, immunological system simulation, process homeostasis…)
    - Etc.

IDS classification

- The fundamental debate between proponents of anomaly detection and proponents of misuse detection
  - Overlap of the regions representing "normal" and "misuse “ activities



IDS classification

- The proponents of anomaly detection assert that the intersection between the two regions is minimal
- The proponents of misuse detection assert that the intersection is quite large, to the point that given the difficulties in characterizing "normal" activity, it is pointless to use anomaly detection

IDS classification

- The solution of this problem is in combining the two detection models
- Although the IDS/IPS manufacturers do not publish the details of their designs, it is quite probable that they combine misuse detection and anomaly detection approach in their solutions
- An example of an IDS that combines these models - *Bro*

Bro fundamentals

- Bro – a network monitoring tool
  - Not (only) an IDS
  - Focuses on application-level semantics
    - The goal is not to alert on occurrences of traffic patterns identical to known attack signatures
    - Instead, *purpose* of traffic patterns is interpreted
      - Example
        - » If an attack is in progress, it is not recognized by exact matching of an attack signature with current traffic
        - » Rather, Bro interprets traffic (e.g. by heuristics) as the one whose purpose is to attack

Bro fundamentals

- Bro rarely analyzes a single packet
- To conclude about purpose of the traffic at hand, several packets are analyzed in order to establish context
- From the received packet sequence, Bro generates *events*
  - Neutral, i.e. not labeled as “good” or “bad”
- Labeling of events is performed in the *Policy script interpreter*

Bro fundamentals

- Several ready-made scripts for determining purpose of the received traffic are present in Bro
  - However, it is supposed that the operator writes the scripts that implement a policy defined in advance

Bro fundamentals

- There is a possibility of using low-level Snort-style signatures in Bro
  - For this, Bro uses an independent signature language
  - When Bro uses signatures, it is converted into a classical misuse detection system
    - However, since other open-source misuse detection systems, such as Snort, are so different from Bro, it is not possible to convert their signatures (for example Snort rules) directly to Bro signatures

Bro fundamentals

- There is no external source of Bro signatures
  - The operator must write all the signatures that will be used in Bro by himself, unlike Snort or Suricata signatures that are written by a specialized organization and maintained regularly
  - This limits the practical use of signatures in Bro
    - There existed a converter of Snort rules to Bro signatures (*Snort2Bro*), but it is now discontinued

Bro fundamentals

- The Bro scripts can be written in such a way that the system behaves like an *anomaly detection system*
- However, Bro is neither misuse nor anomaly detection system by default

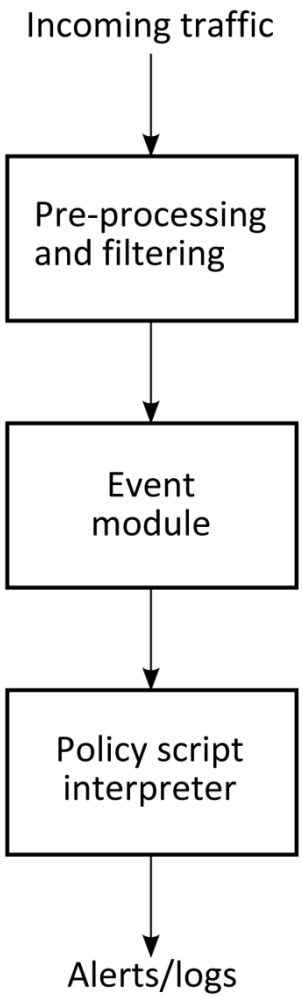
Bro fundamentals

- Bro always logs all the activity, which is useful for forensic analysis
- It also provides support for intrusion prevention
- IDS evasion resistance mechanisms are implemented too

Bro building blocks

- The Bro system is structured in layers
  - Pre-processing and filtering
  - Event engine
  - Policy script interpreter

Bro building blocks



Bro building blocks

- The Pre-processing and filtering stage
  - The lower-most layer
  - Extracts data packets from the network
  - For this purpose, the *libpcap* interface is usually used, but there are other possibilities as well, such as *daq* etc.

Bro building blocks

- The Event engine (1)
  - Generates unbiased interpretation of purpose of the processed packets in the form of special notifications called *events* (1)
    - The event module first performs integrity check of the analyzed packet and if it fails, a special event indicating a problem is generated and the packet is discarded
    - If the integrity of the analyzed packet is in order, the processing continues by detecting the transport protocol used in the packet (e.g. TCP, UDP)

Bro building blocks

- The Event engine (2)
  - Generates unbiased interpretation of purpose of the processed packets in the form of special notifications called *events* (2)
    - Finally, for each supported protocol there is a connection handler
      - Verifies the integrity of the protocol implementation (by means of a checksum) and interprets the purpose of traffic on the semantic level

Bro building blocks

- The Event engine (3)
  - What is monitored is not a single packet, but a *session*
    - Real in the TCP case
    - Virtual in the UDP case
      - UDP session is a set of contiguous packets sent from a determined IP address to another determined IP address
  - The processing of protocol-specific connection handlers ends in invoking handlers to process the payload data

Bro building blocks

- The Event engine (4)
  - The event module generates policy-neutral events interpreting the underlying network activity
  - Examples of events
    - Connection attempt
    - HTTP reply
    - User logged in
    - Etc.

Bro building blocks

- The Event engine (5)
  - The system recognizes more than 300 types of events
  - The complete list of events recognizable by Bro is contained in the file *event.bif.bro*
  - Demo: show the file *event.bif.bro*

    (location: /usr/local/bro/share/bro/base/event.bif.bro)

Bro building blocks

- The Policy script interpreter (1)
  - Processes the stream of events generated in the previous stage
  - Context-aware
    - Should reduce the number of false positives
  - A particular policy, so-called *site policy* is enforced in this module and notifications are generated if the traffic does not conform to it

Bro building blocks

- The Policy script interpreter (2)
  - Performs extensive logging, which serves in forensic analysis
  - Generates alerts via Syslog and sends commands to other applications executed by Bro as a response to suspicious traffic

Bro operation

- Pre-processing and filtering (1)
  - Purpose
    - Reduce the amount of data to be processed by the subsequent stages to a minimum
  - As we go higher up through the Bro layers, the amount of data to be processed is reduced, allowing for more intensive processing per data item

Bro operation

- Pre-processing and filtering (2)
  - The network is tapped in a passive way
  - A copy of all network traffic is sent to the packet capture interface - usually *libpcap*
    - The advantage
      - Compatibility with practically all the operating systems used nowadays
      - Compatibility with standard packet filtering mechanisms, such as BPF (Berkeley Packet Filtering)
    - The disadvantage
      - Not very efficient

Bro operation

- Pre-processing and filtering (3)
  - Filtering packets by means of these mechanisms greatly reduces the amount of traffic processed by the IDS from the very beginning
    - Reduces the false negative rate

Bro operation

- Pre-processing and filtering (4)
  - Bro pre-processing also includes data reduction on the selected packets
    - Only a portion of a packet is necessary to analyze in order to determine the elements of a “session” needed for the context aware intrusion detection
    - The necessary length of a packet for this purpose is called *snapshot length*
      - In general much shorter than a typical length of a packet (1500 bytes).

Bro operation

- Pre-processing and filtering (5)
  - Example
    - The default tcpdump snapshot length is 68 bytes
      - Sufficient to capture link-layer and TCP/IP headers
  - Reducing the snapshot length as much as possible contributes to overall efficiency of the IDS and reduces the number of false negatives

Bro operation

- Pre-processing and filtering (6)
  - In Bro, a single packet is seldom enough for quality estimation of the session properties
  - In order to determine relevant information, the pre-processing stage of Bro collects the control data (SYN, FIN, RST etc.) in several packets

Bro operation

- Pre-processing and filtering (7)
  - Based on these data, the parameters of the session, which are typically the connection start time, duration, participating hosts, ports, the number of bytes sent in each direction, are determined and recorded in a special data structure for further processing in the subsequent stages of Bro

Bro operation

- Event generation (1)
  - Purpose
    - Perform generic analysis of the connection packets
  - Each intercepted packet is checked against the connections established by the event module
  - If such a connection does not exist, a new connection is created, commencing with the current packet

Bro operation

- Event generation (2)
  - Each connection is associated an *analyzer tree*
    - A data structure and a protocol detection mechanism that activates specific analysis modules, called *analyzers* for the connection traffic
    - Dynamically adds and removes these analyzers
    - At activation of each analyzer, all the traffic is assigned to it, no matter how it was processed by the previous analyzers
    - If an analyzer determines that it has been invoked in error, it is deactivated

Bro operation

- Event generation (3)
  - Categories of Bro analyzers (1)
    - Transport layer analyzers - ICMP, TCP, and UDP
    - Application layer analyzers - BitTorrent, DCE RPC, DNS, Finger, FTP, Gnutella, HTTP, Ident, IRC, Login, NCP, NetbiosSSN, NFS, NTP, POP3, Portmapper, Rlogin, RPC, Rsh, SMB, SMTP, SSH, and Telnet

Bro operation

- Event generation (4)
  - Categories of Bro analyzers (2)
    - Support analyzers - Contents, ContentLine, NVT, Zip, Contents DNS, Contents NCP, Contents NetbiosSSN, Contents Rlogin, Contents Rsh, Contents DCE RPC, Contents SMB, Contents RPC, and Contents NFS
    - Other analyzers - File, Backdoor, InterConn, SteppingStone, TCPStats, and ConnSize

Bro operation

- Event generation (5)
  - In addition to the manually written application-layer analyzers, several automatically generated application-layer analyzers exist in the event generation module
    - DHCP BINPAC, DNS TCP BINPAC, DNS UDP BINPAC, SSL, HTTP BINPAC, and SYSLOG BINPAC

Bro operation

- Event generation (6)
  - These analyzers were produced by means of a special protocol parser that holds the same name as the high-level language in which the protocols are described in it - BINPAC
  - It generates C++ code starting from the protocol description in the BINPAC language
  - Then this C++ code is integrated into the Bro system

Bro operation

- Event generation (7)
  - The most complex task of the Bro event generation module is detection of the application protocol
  - In other network intrusion detection systems, detection of application protocol is usually performed in a static way

Bro operation

- Event generation (8)
  - Example (1)
    - In Snort, a static list of http ports used in protocol detection is provided in the configuration file *snort.conf*
    - The problem with this approach is in the fact that malicious packets tend not to use traditional ports for their activities
    - This forces the IDS designers to constantly expand the static list of ports by adding ports on which they have observed certain type of traffic

Bro operation

- Event generation (9)
  - Example (2)
    - Snort checks more than 40 ports for http traffic
    - The consequence is reduced overall efficiency of the IDS and the possibility of IDS evasion by using ports not in the current list for delivering malicious traffic
    - Demo: Display the contents of the file *snort.conf* /etc/snort/snort.conf

Bro operation

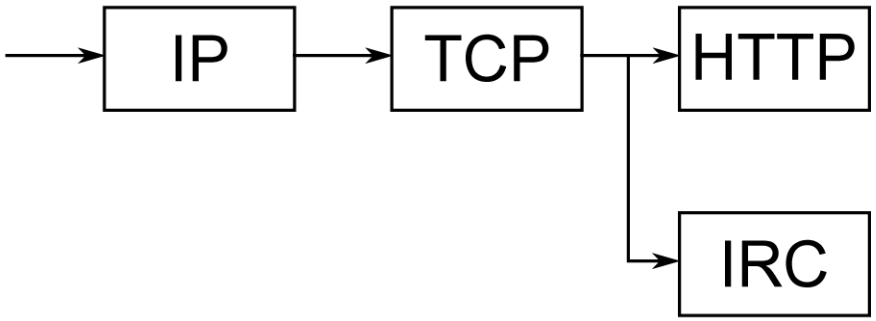
- Event generation (10)
  - In addition, it is possible to use non-standard services on the ports traditionally dedicated to certain services
  - Example
    - The use of IRC on a port traditionally used for http traffic

Bro operation

- Event generation (11)
  - The power of Bro lies in dynamic protocol analysis, performed in the event generation module
  - Bro maintains a tree-like data structure called *data path* for every session tracked
  - The nature of this data structure is dynamic, i.e. it may change during the analysis

Bro operation

- Event generation (12)
  - The tree-like data structure represents the data path through the analyzers
  - Example – protocol analyzer tree in Bro

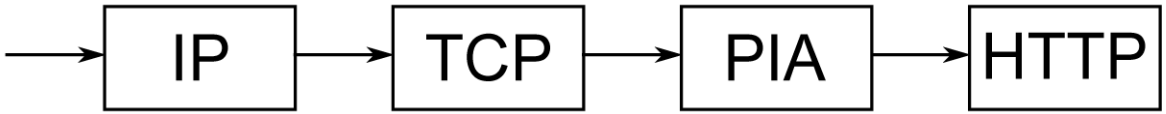


Bro operation

- Event generation (13)
  - Initially, the analyzer tree of a new session consists of the low-level protocol analyzers
  - Example
    - If the first packet of the session uses the TCP protocol, the analyzer tree will initially consist of an IP analyzer followed by a TCP analyzer

Bro operation

- Event generation (14)
  - To recognize the higher level protocols, a set of algorithms and heuristics is used
  - They are implemented in a *Protocol Identification Analyzer* (*PIA*)
  - PIA is inserted into the analyzer tree after the TCP or UDP analyzer



Bro operation

- Event generation (15)
  - After detecting an application protocol, a PIA instantiates a descendant analyzer for it and forwards the data to that particular analyzer
  - But the PIA is not dismissed after this action
    - It continues applying its algorithms and heuristics on the received traffic
    - If it finds another match for a protocol, it instantiates additional or alternative analyzers

Bro operation

- Event generation (16)
  - During execution of the analyzers instantiated by a PIA, it may happen that they are not capable of determining the details of the protocols corresponding to their output data
  - In that case, they may decide to instantiate another PIA in order to solve this problem
  - Thus, an analyzer tree may contain more than one PIA

Bro operation

- Event generation (17)
  - Design of the event module (1)
    - Basic design criteria (1)
      - The need for different approaches in protocol detection
        - » The port number is obviously not enough and because of that the use of signatures, heuristics and statistical methods is also needed
      - Flexible customization of the analyzers
        - » The configuration of the analyzers must conform to the criteria that led to their activation (port numbers, patterns for application-layer protocol detection, custom decision functions, etc.)

Bro operation

- Event generation (18)
  - Design of the event module (2)
    - Basic design criteria (2)
      - Dynamic activation/deactivation of the analyzers
        - » The analyzers might be activated at any time while the connection is active
        - » If during the analysis it turns out that an analyzer was invoked in error, it should be deactivated

Bro operation

- Event generation (19)
  - Design of the event module (3)
    - Basic design criteria (3)
      - Performance requirements
        - » Since IDS must operate in high-speed networks in real time, the performance of the analyzers must be as high as possible
        - » It is, however, known that the cost of dynamic analysis of protocols is higher than the cost of static analysis
        - » Because of that, it is required that static analysis remains as a possibility, if performance of protocol analysis becomes critical for the efficient operation of the IDS

Bro operation

- Event generation (20)
  - Design of the event module (4)
    - Basic design criteria (4)
      - Possibility for multiple instances of analyzers
        - » The situation may dictate that multiple analyzers are applied on the same data
      - Capability of analyzing tunneled connections
        - » The dynamic analysis should be available for tunneling protocols as well
        - » This may be achieved through allowing more PIAs in the analyzer tree

Bro operation

- Event generation (21)
  - Design of the event module (5)
    - Basic design criteria (5)
      - The need for standardized interfaces for the analyzers
        - » It is necessary to connect different analyzers in a serial manner, according to the protocol stack (for example, IP-TCP-application)
        - » Each analyzer then processes input provided from the previous analyzer and produces output that is going to be utilized by the next analyzer in the row
        - » Because of that, the design of common interfaces is very important for efficient operation of an analyzer tree

Bro operation

- Event generation (22)
  - Design of the event module (6)
    - Basic design criteria (6)
      - The need for extension of the analyzers
        - » Some modules and/or built-in functions could be used by various analyzers in order to carry out common tasks

Bro operation

- Event generation (23)
  - Design of the event module (7)
    - Techniques for identifying protocols (1)
      - Behavioral analysis
        - » A form of statistical identification
        - » Such an analysis is based on probability study that creates a protocol profile that includes probabilities that certain users, ports or servers will be included in the protocol data
        - » Various machine learning techniques can also be used for this purpose

Bro operation

- Event generation (24)
  - Design of the event module (8)
    - Techniques for identifying protocols (2)
      - Protocol structure based detection
        - » Based on a careful study of the header and data structure of the currently analyzed protocol in order to conclude about what the next protocol in the protocol stack will be

Bro operation

- Event generation (25)
  - Design of the event module (9)
    - Techniques for identifying protocols (3)
      - Signature-based detection
        - » The most often used technique, since the misuse detection systems already implement various signature matching techniques
        - » In such a system, all the supported protocols are defined through a set of patterns that identify them
        - » These patterns are then matched against every connection

Bro operation

- Event generation (26)
  - Design of the event module (10)
    - Techniques for identifying protocols (4)
      - A combination of these techniques is also possible
      - Example
        - » It is possible to use statistical methods to identify some application protocols and signatures to identify others

Bro operation

- Event generation (27)
  - Design of the event module (11)
    - In most cases, it is necessary to buffer data from the incoming packets in order to perform reassembly later (for example, HTTP reassembly)
    - This is implemented in PIA
      - Buffers the beginning of each session up to a configurable threshold (by default 4 KB)
      - The buffering ensures that the protocol analyzer activated by PIA starts the analysis from the beginning of the session, provided a buffer overflow does not take place

Bro operation

- Event generation (28)
  - Design of the event module (12)
    - If a PIA decides that an analyzer was invoked in error and that it should start a new one, the buffering mechanism ensures that the whole session will be available to the new analyzer, even though the new analyzer is started in the middle of the session

Bro operation

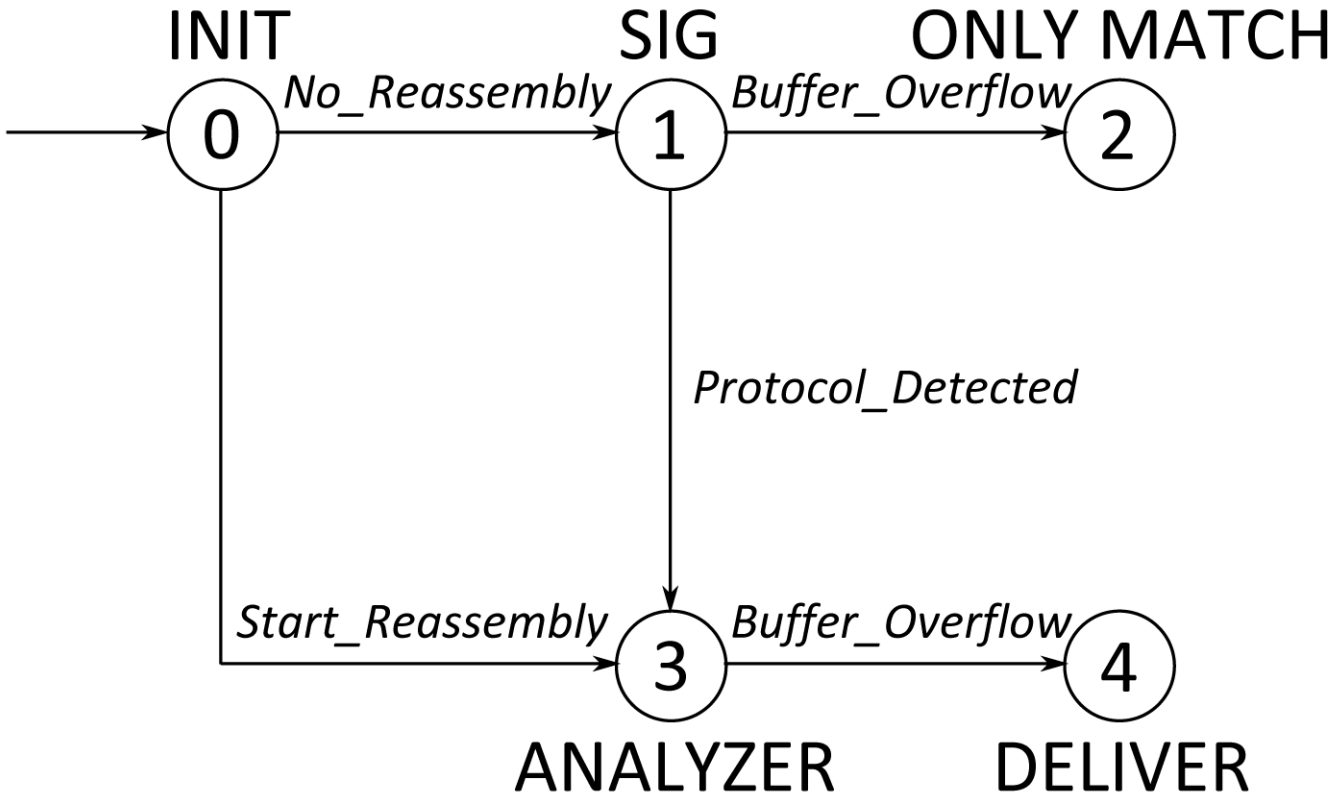
- Event generation (29)
  - PIA structure and operation (1)
    - A PIA can be viewed and analyzed as a finite state machine
    - Possible states of a PIA
      - The INIT state
      - The SIG state
      - The ANALYZER state
      - The ONLY MATCH state
      - The DELIVER state

Bro operation

- Event generation (30)
  - PIA structure and operation (2)
    - Transitional events in a PIA
      - No Reassembly
      - Start Reassembly
      - Buffer Overflow
      - Protocol Detected

Bro operation

- Event generation (31)
  - PIA structure and operation (3)



Bro operation

- Event generation (32)
  - PIA structure and operation (4)
    - The INIT state (1)
      - The state of a PIA when it is instantiated
      - In the INIT state, the PIA retrieves the data from the current analyzer tree about the session (for example, the destination port)
      - The initial configuration of analyzers is decided in that state
      - Then the PIA decides whether there is a need for session reassembly or not
      - If there are active analyzers for the particular session, then it must be reassembled

Bro operation

- Event generation (33)
  - PIA structure and operation (5)
    - The INIT state (2)
      - The initial configuration of Bro determines for which ports the sessions will always be reassembled, even if there are no active analyzers expecting the session data
      - It is also possible to configure Bro to reassemble all sessions
      - If session reassembly is not needed, the PIA finite state machine makes a transition to the SIG state
      - Otherwise, the PIA makes a transition to the ANALYZER state
      - Session reassemblers are implemented in Bro as separate auxiliary analyzers

Bro operation

- Event generation (34)
  - PIA structure and operation (6)
    - The SIG state (1)
      - In the SIG state, the PIA tries to match the session data with protocols, for which it has signatures defined
      - The session is not reassembled and consequently no analyzers are activated until a match with a signature is found
      - Buffering of the session data takes place, however (up to 4 KB of data, by default)

Bro operation

- Event generation (35)
  - PIA structure and operation (7)
    - The SIG state (2)
      - If no signatures are matched in the SIG state, the buffer is filled until the Buffer Overflow event occurs
      - Then the PIA makes a transition to the ONLY MATCH state
      - Otherwise, if a signature match is detected (the event Protocol Detected takes place), the PIA makes a transition to the ANALYZER state

Bro operation

- Event generation (36)
  - PIA structure and operation (8)
    - The ANALYZER state (1)
      - In the ANALYZER state, the PIA expects reassembled application data
      - The activities of the PIA in this state are matching, buffering, delivering and check for buffer overflow

Bro operation

- Event generation (37)
  - PIA structure and operation (9)
    - The ANALYZER state (2)
      - Matching checks, by means of the built-in pattern matching module, whether a pattern identifying another protocol is found in the reassembled data
      - If so, the corresponding analyzer is instantiated and the data from the buffer accumulated during the analysis is replayed to the new analyzer
      - This is so-called *matching with a positive pattern*

Bro operation

- Event generation (38)
  - PIA structure and operation (10)
    - The ANALYZER state (3)
      - Bro also performs *matching with negative patterns* in this state
        - » On such a match, the corresponding analyzer is shut down, which means that the analysis performed by the PIA concludes that the supposition about the protocol by which it came to the ANALYZER state was wrong and should be changed

Bro operation

- Event generation (39)
  - PIA structure and operation (11)
    - The ANALYZER state (4)
      - The buffering activity collects data, their size and direction in a special data structure
      - Its size is limited to 4 KB by default, but this value can be overridden
      - If the size of all objects in the buffer overcomes the limit, a Buffer Overflow event takes place
      - Otherwise, the data are delivered to all the existing successors of the PIA by means of the delivering activity

Bro operation

- Event generation (40)
  - PIA structure and operation (12)
    - The ONLY MATCH state
      - If the PIA in the SIG state fails to find a match for any protocol known by Bro before the Buffer Overflow takes place, the PIA makes a transition to the ONLY MATCH state
      - In this state, the non-reassembled data are immediately matched (i.e. without buffering) against protocol signatures
      - A match with a negative pattern stops the corresponding analyzer
      - The ONLY MATCH state of the PIA is final - The PIA remains in that state until the end of the session

Bro operation

- Event generation (41)
  - PIA structure and operation (13)
    - The DELIVER state
      - A Buffer Overflow event in the ANALYZER state triggers a transition of the PIA finite state machine to another final state, the DELIVER state
      - The incoming data are only matched and delivered to succeeding analyzers
      - A match with a negative pattern also causes shutting down of the corresponding analyzer

Bro operation

- Event generation (42)
  - Analyzer structure and operation (1)
    - For each protocol recognizable by Bro there is a separate analyzer
      - Checks whether the traffic conforms to a particular protocol
    - The input format varies for different analyzers
      - Depends on whether the protocol operates with segments or packets, whether reassembly takes place or not etc.
    - Because of that, all analyzers in Bro are capable of receiving and outputting packet/segment and stream (reassembled) information

Bro operation

- Event generation (43)
  - Analyzer structure and operation (2)
    - Example
      - A TCP analyzer requires reassembled information, i.e. stream input
      - An IP analyzer requires segment input

Bro operation

- Event generation (44)
  - Analyzer structure and operation (3)
    - The analyzers also have to be capable of pre-formatting or normalizing their input
    - Example
      - The analyzers such as Telnet, HTTP, FTP, or SMTP need stream-to-line conversion of input data, i.e. the input stream has to be split into lines
      - For such a conversion, an auxiliary analyzer is used

Bro operation

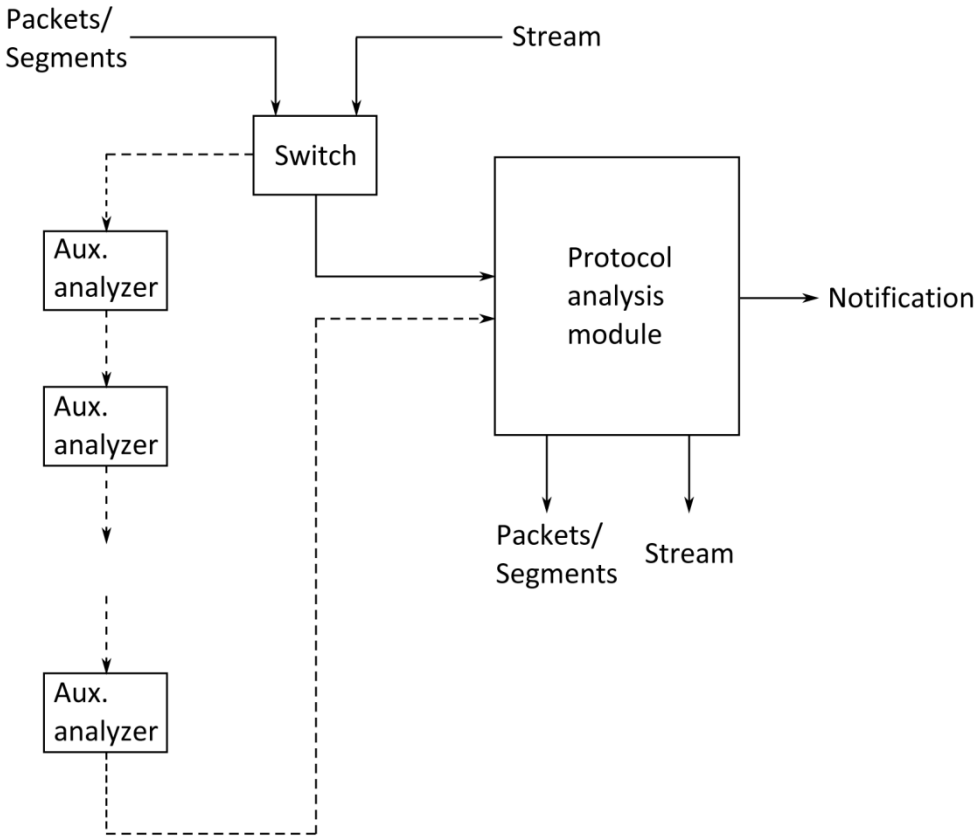
- Event generation (45)
  - Analyzer structure and operation (4)
    - The analyzers are serially coupled
      - This means that the information from an auxiliary analyzer is passed to the next auxiliary analyzer in the series until the correct formatting is achieved
      - Then the formatted information is sent back to the starting analyzer, which is now capable of performing the analysis

Bro operation

- Event generation (46)
  - Analyzer structure and operation (5)
    - Each analyzer in Bro is capable of detecting the situation where the analyzed traffic does not conform to the respective protocol
    - In that case, the analyzer replies to the caller with the signal NACK, causing the previous analyzer to shut it down

Bro operation

- Event generation (47)
  - Analyzer structure and operation (6)



Bro operation

- Event generation (48)
  - Analyzer structure and operation (7)
    - Each analyzer has two inputs
      - Enables seamless processing of both packet/segment and stream (already reassembled) data
    - If any of the auxiliary analyzers are to be used, the built-in switch dispatches the input data to those analyzers
    - Otherwise, the data are sent directly to the protocol analysis module

Bro operation

- Event generation (49)
  - Analyzer structure and operation (8)
    - If some auxiliary analyzers are to be employed, upon serial processing by those analyzers, the data (eventually pre-formatted or changed in some other way in order to be processed properly) are sent to the protocol analysis module

Bro operation

- Event generation (50)
  - Analyzer structure and operation (9)
    - If any irregularities are detected in the protocol analysis module, a notification is generated and sent to the alert subsystem
      - There, depending on the severity of such an irregularity and the current site security policy, a message could be sent to the operator
      - Such a message in the Bro terminology is called a *notice*

Bro operation

- Event generation (51)
  - Analyzer structure and operation (10)
    - If there are no irregularities in the input data, the protocol analysis module forwards the data to the next analyzer in the row
    - As with the input data, the output data are also sent in the packet/segment or stream format

Bro operation

- Event generation (52)
  - Solving problems in event generation (1)
    - To ensure efficient operation of the Bro event module, it is necessary to minimize the numbers of false negatives and false positives in protocol recognition
    - Since protocol detection and verification are two phases of this process, each of these phases has to be configured properly

Bro operation

- Event generation (53)
  - Solving problems in event generation (2)
    - The detection phase accuracy and efficiency are determined by the choice of appropriate protocol signatures used to detect potential application-layer protocols in input data
    - In this phase, it is allowed to have less precise signatures
      - Reduces the number of false negatives
      - The inevitable increase in the number of false positives is compensated by activating the second phase - protocol verification in the analyzers

Bro operation

- Event generation (54)
  - Solving problems in event generation (3)
    - A two-phase protocol recognition mechanism is resource intensive
      - May have negative effect on overall efficiency of the Bro IDS if the second phase of the recognition algorithm - the verification phase in the analyzers - is used too extensively
    - In addition, an attacker may exploit this two-phase procedure and craft traffic that deliberately forces intensive use of the analyzer verification

Bro operation

- Event generation (55)
  - Solving problems in event generation (4)
    - To achieve balance in the use of the two phases of the protocol recognition process, Bro uses bidirectional signatures for protocol detection (the first phase)
    - This means that a match means that both endpoints of a session are supposed to use the same protocol

Bro operation

- Event generation (56)
  - Solving problems in event generation (5)
    - Then an attacker that controls one side of the session cannot force Bro event module to activate the corresponding analyzer and thus increase resource consumption of Bro
    - In addition, before activating the second phase, Bro parses the beginning of the contents of the traffic in both directions
      - The corresponding analyzer is activated only if this parsing process detects meaningful information exchange between the session endpoints

Bro operation

- Event generation (57)
  - Solving problems in event generation (6)
    - Event generation timing is also one of the important parameters of Bro quality
    - If use of certain applications (and consequently of the corresponding protocols) violates the site security policy, it should be reported before the application in question causes any harm

Bro operation

- Event generation (58)
  - Solving problems in event generation (7)
    - The first phase of the application protocol detection procedure, the signature matching, is completed quickly
    - However, to be certain about the verification, the second phase is needed, in which one should wait until the session ends

Bro operation

- Event generation (59)
  - Solving problems in event generation (8)
    - Waiting until the end of session is unacceptable, since during the very session the malicious activity intended by the use of the forbidden application protocol could be carried out
    - Because of that, any analyzer in Bro supposes that if the beginning of the session uses an application protocol, the rest of the session also uses the same protocol

Bro operation

- Event generation (60)
  - Solving problems in event generation (9)
    - Bro uses two thresholds
      - The exchange of the given volume of initial information between the session endpoints
      - The amount of time that is used for the session beginning
    - Both thresholds are configurable
    - It is evident that such kind of Bro operation is vulnerable to attacks that would exploit the values of those thresholds, but that is the price that we must pay for alerting on time when an attack occurs

Bro operation

- Event generation (61)
  - Solving problems in event generation (10)
    - An analyzer may detect that it has been invoked in error, i.e. that the protocol used does not conform to the analyzer invoked
    - Since this may be a result of a random communication error, it is not a good idea to shut down the analyzer immediately
    - On the other hand, waiting for too long for the resynchronization of the session may lead to waisting of resources

Bro operation

- Event generation (62)
  - Solving problems in event generation (11)
    - The solution
      - The sessions in Bro are rejected if they violate a set of basic protocol properties
    - But in that case, an attacker could exploit this fact and generate traffic that does not violate the set of protocol specifications analyzed by Bro, but still be illegal

Bro operation

- Event generation (63)
  - Solving problems in event generation (12)
    - To reduce the chance of success of such an attack, an analyzer in Bro never shuts down itself automatically
    - Instead, in the case of severe protocol violations, Bro generates an event that is to be processed by the policy script interpreter
    - It is then up to the user to say what is to be done with such traffic

Policy script interpreting

- The main characteristic of Bro
  - Separation of its surveillance mechanism implemented in the pre-processing and event generation stages from a site policy for processing the generated events
- The site-specific security policy defines the interpretation of these events, labeling them as normal, abnormal, suspicious and so on

Policy script interpreting

- Once an event is triggered in the event generation stage, it is passed to the *policy layer* that processes these events
- The policy layer defines the policy configuration that embodies the site security policy
  - Realized through scripts written in the specific Bro scripting language

Policy script interpreting

- The relevance of an event varies depending on the site (or context)
  - Using a site specific policy offers a possibility of reflecting this relevance in the corresponding scripts
  - Reduces the number of false positives
- The scripts realize event handlers
  - Process the generated events according to the site specific security policy

Policy script interpreting

- There is a default set of scripts that are loaded when Bro starts
- These scripts can
  - Report events related with use of various protocols (e.g. http, ssh, ssl, irc, ftp and so on)
  - Report use of software embedded in connections
    - Example - use of Nmap port scanning tool embedded in HTTP requests, etc.

Policy script interpreting

- In addition to loading the default scripts, there are many ready-made scripts that can be loaded upon request
- It is also possible to run Bro without using the default scripts
  - Then the user has full control over behavior of the system
  - He can load only the scripts necessary for detection of suspicious traffic at hand

Policy script interpreting

- The Bro script set structure (1)
  - The Bro script set that defines its functionality is divided into three categories: *Base*, *Policy* and *Site*
  - Each of these categories contains a corresponding set of scripts that can be loaded when Bro starts
  - Physically, the scripts are located in the separate directories, whose names are the same as the names of the categories

Policy script interpreting

- The Bro script set structure (2)
  - Demo: Display the contents of the directory /usr/local/bro/share/bro

Policy script interpreting

- The Bro script set structure (3)
  - The Base category scripts are loaded by default when Bro starts
    - These scripts define essential functionality of Bro, which includes the Bro built-in function that defines events (event.bif.bro), some data types, utility functions as well as basic frameworks (e.g. *notice*) and events related with the supported protocols (http, ftp, irc, smtp, ssh, ssl, syslog, etc.)
    - Demo: Display the contents of the directory /usr/local/bro/share/bro/base

Policy script interpreting

- The Bro script set structure (4)
  - The Policy category scripts are not completely loaded at the start-up time
    - In general, users have to define whether they want to use some of them
    - These scripts include those that perform a deeper inspection of packets that convey specific protocol traffic, as well as more frameworks
    - Demo: Display the contents of the directory /usr/local/bro/share/bro/policy

Policy script interpreting

- The Bro script set structure (5)
  - The Site category defines the local security policy
    - In the script *local.bro*, this policy is defined by specifying the scripts from other categories that are to be loaded at the start-up time
    - Users can edit this script according to their needs
    - In addition, they can write their own scripts thus effectively defining the site security policy
    - In such a way, the user can precisely define what is considered an attack on the particular network

Policy script interpreting

- The Bro script set structure (6)
  - Demo
    - Display the contents of the directory /usr/local/bro/share/bro/site
    - Display the file local.bro

Policy script interpreting

- The Bro script set structure (7)
  - The contents of each script loaded in Bro defines what action is to be performed related to an event discovered by the Event module
  - Most often these scripts analyze the events thoroughly, by extracting the necessary information
    - This information can then be used by other scripts, typically by those that raise the alerts

Policy script interpreting

- The Bro script set structure (8)
- Bro activates some of the alert-raising scripts at the start-up time by default
  - Most of such scripts belong to the Site category, which means that they are user-defined

Policy script interpreting

- The Bro script set structure (9)
  - The Base script set
    - The Base script set consists of four subsets: *Frameworks*, *Misc*, *Protocols*, and *Utils*
    - In addition, this set contains a set of built-in functions (bif) that provide support to the system by implementing general programming algorithms, string processing, mathematical functions, type declarations, etc.

Policy script interpreting

- The Bro script set structure (10)
  - The Framework script subset
    - The Framework script subset consists of the following script groups (frameworks): Cluster, Communication, Control, DPD, Intel, Logging, Metrics, Notice, Packet filter, Reporter, Signatures, Software, etc.
    - Demo: Display the contents of the directory /usr/local/bro/share/bro/base/frameworks
    - Each group contains a script named *main.bro*
      - This script is sometimes accompanied with additional auxiliary scripts

Policy script interpreting

- The Bro script set structure (11)
  - Cluster
    - The Bro network security monitor is capable of using several Bro sensors combined in a distributed architecture, which is called *cluster* in Bro terminology
    - The Cluster script group provides support for establishing and controlling a cluster of Bro instances

Policy script interpreting

- The Bro script set structure (12)
  - Communication (1)
    - Provides support for communication between several instances of Bro, as well as communication between various tools written in the C programming language and Bro
    - The different instances of Bro can share their states and transfer events between them

Policy script interpreting

- The Bro script set structure (13)
  - Communication (2)
    - Communication between C tools and Bro is carried out by means of a special library *Bro Client Communications Library – Broccoli*
      - Can be used for transfer of data to and from Bro
        - » Example - SYSLOG events can be fed into Bro by means of Broccoli

Policy script interpreting

- The Bro script set structure (14)
  - Control
    - Provides support for commands that can be sent remotely at runtime to modify a running Bro instance or collect information from a running Bro instance
    - The following commands are supported: id_value, peer_status, net_stats, configuration_update, and shutdown

Policy script interpreting

- The Bro script set structure (15)
  - DPD (1)
    - The *Dynamic Protocol Detection* - DPD framework represents support to the protocol detection algorithm that does not use port numbers
    - The main script in this group is capable of disabling the analyzers in the event module if protocol violations occur
    - The main script is supported with a set of protocol recognition signatures contained in the file *dpd.sig*

Policy script interpreting

- The Bro script set structure (16)
  - DPD (2)
    - Demo: Display the file *dpd.sig*

      (location:
      /usr/local/bro/share/bro/base/frameworks/dpd/dpd.sig)

Policy script interpreting

- The Bro script set structure (17)
  - DPD (3)
    - Example (1)

```
signature dpd_http_client {
    ip-proto == tcp
    payload /^[[:space:]]*(GET|HEAD|POST)[[:space:]]*/
    tcp-state originator
}
```

Policy script interpreting

- The Bro script set structure (18)
  - DPD (4)
    - Example (2)

```
signature dpd_http_server {
    ip-proto == tcp
    payload /^HTTP\/[0-9]/
    tcp-state responder
    requires-reverse-signature dpd_http_client
    enable "http"
}
```

Policy script interpreting

- The Bro script set structure (19)
  - DPD (5)
    - Example (3)
      - This signature defines an HTTP client and an HTTP server
      - The HTTP server requires an HTTP client in order for that signature to be matched
      - The protocol is recognized by recognizing some elements in the payloads of the packets
      - These elements are given in the regular expression form
      - The keyword *enable* activates the corresponding analyzer in the event module

Policy script interpreting

- The Bro script set structure (20)
  - Intel
    - Provides support for storing and querying IP addresses, string and numerical data
    - Also capable of associating metadata (such as tags) with the intelligence data
    - Examples of strings recognized in IP addresses and associated data are url, user name, file name, etc.
    - The tags that can be associated to these data are malicious, sensitive, etc.

Policy script interpreting

- The Bro script set structure (21)
  - Logging
    - Provides the logging interface for Bro
    - Defines logging filters, local and remote logging, default logging device, the contents of the logging stream and so on
    - Also provides support for making secure copies of log streams and secure transfer of logs

Policy script interpreting

- The Bro script set structure (22)
  - Metrics
    - Provides support for counting and measuring data
    - Closely related to the Notice script group, since many notices (the messages to the operator in Bro) are based on some counters overcoming some thresholds

Policy script interpreting

- The Bro script set structure (23)
  - Notice (1)
    - The Notice framework enables Bro to send *notices*
      - Messages to the operator about odd or potentially malicious traffic
    - Defines logging streams for notices
      - The primary logging stream is LOG, but the notices are very often escalated to alerts and because of that the ALARM_LOG is also present as a logging stream
    - There is also a policy auditing log POLICY_LOG
      - Records what the current notice policy is at Bro initialization time

Policy script interpreting

- The Bro script set structure (24)
  - Notice (2)
    - The Notice framework also defines actions to be undertaken upon generating notices (ACTION_NONE, ACTION_LOG, ACTION_EMAIL, ACTION_ALARM, ACTION_NO_SUPRESS (i.e. keep generating repeated notices about the same event))
    - It counts the number of notices and defines the report fields in the records of the notice log
    - It also defines which notice types are to be ignored, sent by E-mail, alarmed etc.

Policy script interpreting

- The Bro script set structure (25)
  - Notice (3)
    - In addition, this framework defines which elements constitute the notice policy
    - Priorities of the notices are defined as well as many other details related to sending notices
    - Several supporting scripts are also present, besides the main script
      - They define the Notice framework in the cluster environment, post-processing of notices, actions related to non-standard (*weird*) traffic etc.

Policy script interpreting

- The Bro script set structure (26)
  - Notice (4)
    - Demo: Display the file *main.bro* in the *notice* framework

      (location: /usr/local/bro/share/bro/base/frameworks/notice/main.bro)

Policy script interpreting

- The Bro script set structure (27)
  - Packet filter (1)
    - This framework defines settings for the Berkeley Packet Filter (BPF) used by Bro
    - The default is that no traffic is filtered out
    - This default filter can be overridden by the command line and the capture_filters and restrict_filters variables
    - The command line filter takes precedence over all the other filters

Policy script interpreting

- The Bro script set structure (28)
  - Packet filter (2)
    - The packet filter used in Bro is particularly useful if packet drop takes place
    - The number of dropped packets is reported by means of the script *netstats.bro*

Policy script interpreting

- The Bro script set structure (29)
  - Reporter (1)
    - Provides support for filtering internal messages, warnings and errors in Bro
    - Its role is to prevent that internal messages end up in standard reporting channels and thus confuse the operator
    - Internal messages are instead logged in a separate log file *reporter.log*
    - These messages are very useful at new script debug time

Policy script interpreting

- The Bro script set structure (30)
  - Reporter (2)
    - There are three types of messages: INFO, WARNING, and ERROR
    - An error message is not a fatal error, i.e. it does not terminate Bro execution

Policy script interpreting

- The Bro script set structure (31)
  - Signatures
    - Bro also supports classical misuse-based intrusion detection, through its own signature framework
    - The Signatures framework in the Base script set provides support for this type of intrusion detection
    - It defines types of signatures, actions associated with signatures, report fields written in the corresponding log, etc.

Policy script interpreting

- The Bro script set structure (32)
  - Software
    - Provides support for software version detection and parsing
    - However, it does not detect software by itself
    - Instead, it uses information provided by protocol-specific scripts
    - The framework records the version number of the discovered software, defines the report fields for the corresponding log file and so on

Policy script interpreting

- The Bro script set structure (33)
  - The Misc script subset
    - This script subset currently contains fingerprints of various operating systems that serve for identifying them on the basis of the incoming SYN packets
      - Window size, initial TTL, status of the don't fragment bit, SYN packet size, etc. serve for this purpose

Policy script interpreting

- The Bro script set structure (34)
  - The Protocols script subset
    - Provides support for detecting events related with various application protocols
    - Currently, the support is provided for the following protocols: DNS, FTP, HTTP, IRC, SMTP, SSH, SSL, SYSLOG, etc.
    - In addition, there is a special script group CONN that extracts relevant information about tcp and udp sessions, as well as icmp type and code information

Policy script interpreting

- The Bro script set structure (35)
  - CONN (1)
    - Provides support for tracking and logging of general information related with TCP, UDP, and ICMP traffic
    - A *connection* with TCP or UDP is considered a sequence of packets exchanged between a determined source host/port and a destination host/port
    - Regarding ICMP traffic, the *type* and *code* fields are extracted from ICMP packets

Policy script interpreting

- The Bro script set structure (36)
  - CONN (2)
    - The contents of the packets can be extracted too, by means of the script *contents.bro*, but this is not done by default
    - Special processing is provided for traffic with long delays between packets by means of the script *inactivity.bro*

Policy script interpreting

- The Bro script set structure (37)
  - DNS
    - Tracks and logs DNS queries and responses
    - Report fields are defined for logging, the status of DNS queries, ports related with the DNS protocol and so on
    - The types, errors and fields for analysis of DNS data are provided in a separate script *consts.bro*

Policy script interpreting

- The Bro script set structure (38)
  - FTP
    - Provides support for logging FTP commands together with the corresponding metadata
    - Defines the report fields for logging
    - Defines FTP-related ports
    - Also defines the FTP commands and provides functions for extracting information from them
    - File extraction support is provided by means of a separate script *file-extract.bro*

Policy script interpreting

- The Bro script set structure (39)
  - HTTP (1)
    - Provides support for HTTP protocol analysis in Bro
    - Logs request/response pairs and the relevant metadata
    - It is possible to define types of attacks on HTTP that are to be logged
    - The record fields for logging are defined, as well as possible states of an HTTP connection, HTTP headers, the HTTP-related ports with distinguished likely-server ports, etc.

Policy script interpreting

- The Bro script set structure (40)
  - HTTP (2)
    - The functions that carry out HTTP data extraction are also provided
    - In addition to the *main.bro* script that performs basic analysis, several auxiliary scripts provide support for per-file extracting items from HTTP traffic, calculating hash values for HTTP body transfers and identifying file types in HTTP response bodies

Policy script interpreting

- The Bro script set structure (41)
  - HTTP (3)
    - Signatures for identifying application types are given in the file *file-ident.sig*
    - Various utilities used in the scripts are provided in a special script *utils.bro*
    - Demo: Display the contents of the file *main.bro*
      - (location: /usr/local/bro/share/bro/base/protocols/http/main.bro)

Policy script interpreting

- The Bro script set structure (42)
  - IRC
    - Provides the IRC analysis support
    - IRC commands together with the associated responses and additional metadata about the connection are logged
    - The report fields for logging are defined, as well as the IRC-related ports
    - The functions to extract data are also provided
    - Functionality to analyze Direct Client Connection (DCC) file transfers is provided in the script *dcc-send.bro*

Policy script interpreting

- The Bro script set structure (43)
  - SMTP
    - Provides support for analysis of E-mail traffic that uses the SMTP protocol
    - Report fields for logging are defined, as well as the possible states of a connection
    - The SMTP-related ports are also defined
    - The functions that carry out information extraction are provided
    - Analysis and logging of MIME entities is performed by means of a separate script *entities.bro*

Policy script interpreting

- The Bro script set structure (44)
  - SSH (1)
    - Provides a heuristic to determine whether an SSH connection was successful or not
    - The heuristic uses the size of data returned from the server as an indicator
    - The report fields for logging are also defined in this script

Policy script interpreting

- The Bro script set structure (45)
  - SSH (2)
    - The threshold for the returned data size is set to 5500 bytes by default
    - If the data size overcomes this threshold the SSH connection is considered successful
    - The SSH-related ports are defined and the functions to extract data are provided

Policy script interpreting

- The Bro script set structure (46)
  - SSL (1)
    - Provides support for analysis of the SSL/TLS handshaking and the establishment of the enciphered connection
    - Defines the report fields for logging
    - Defines the root Certification Authority (CA) and the SSL-related ports
    - Also provides the functions that extract data

Policy script interpreting

- The Bro script set structure (47)
  - SSL (2)
    - There is a special function that de-activates the SSL analysis after the SSL connection is established
      - This function increases performance of further analysis, since the resources used for SSL connection establishment analysis are freed by means of it
    - A list of root CA is provided in the file *mozilla-ca-list.bro*
    - The numerical and string code constants necessary for SSL connection establishment analysis are provided in the script *consts.bro*

Policy script interpreting

- The Bro script set structure (48)
  - SYSLOG
    - Provides support for logging SYSLOG messages in Bro
    - The report fields for logging are provided as well as the SYSLOG-related ports
    - Every SYSLOG message is analyzed as an event, for which various parameters are defined in the script *main.bro*
    - The numerical and string constants necessary for SYSLOG data extraction are given in the file *consts.bro*

Policy script interpreting

- The Bro script set structure (49)
  - The Utils script subset (1)
    - Implements algorithms that other scripts need to realize their tasks
    - The implemented functions (1)
      - Parse and manipulate IP addresses
      - Generate ASCII strings from connection identifiers
      - Determine whether a connection is unidirectional or bidirectional, inbound or outbound
      - Generate various file names
      - Extract integers from strings

Policy script interpreting

- The Bro script set structure (50)
  - The Utils script subset (2)
    - The implemented functions (2)
      - Parse and manipulate UNIX-style paths and directories
      - Create and manipulate patterns (regular expressions)
      - Perform small string analysis, etc.
    - In addition, in the file *site.bro*, it is defined which networks are local and what servers run particular services

Policy script interpreting

- The Bro script set structure (51)
  - The Policy script set
    - The Policy script set consists of five subsets: *Frameworks*, *Integration*, *Misc*, *Protocols*, and *Tuning*
    - All these subsets contain auxiliary scripts that are not necessarily loaded to Bro by default
    - The operator can include them by specifying their names in the Site policy file *load.bro*

Policy script interpreting

- The Bro script set structure (52)
  - The Framework script subset
    - Consists of the following script groups (frameworks): Communication, Control, DPD, Metrics, Signatures, and Software
    - The Policy framework script groups do not contain scripts named main.bro

Policy script interpreting

- The Bro script set structure (53)
  - Communication
    - Facilitates communication between various Bro instances
    - Implements a listener, a script that enables Bro to wait other Bro instances to connect to it.

Policy script interpreting

- The Bro script set structure (54)
  - Control
    - The scripts *controller.bro* and *controllee.bro* constitute this script group
    - The script controller.bro implements the interface intended to control a remote Bro instance and then shut it down
    - The script controllee.bro is loaded by the Bro process that is planned to receive control commands from another Bro process

Policy script interpreting

- The Bro script set structure (55)
  - DPD (1)
    - Consists of the scripts *detect-protocols.bro* and *packet-segment-logging.bro*
    - The script detect-protocols.bro finds protocols that run on non-standard ports
    - It first defines a list of ports on which detection of a new protocol in the specified direction is irrelevant

Policy script interpreting

- The Bro script set structure (56)
  - DPD (2)
    - Example - if HTTP outgoing traffic is detected on the port 81, it is considered benign
    - All other traffic detected on that port will be considered suspicious and a notice will be raised
    - The notices can be Protocol_Found or Server_Found

Policy script interpreting

- The Bro script set structure (57)
  - DPD (3)
    - A connection is considered to use a protocol *X* if it is still active after minimum 30 sec, or if the minimum traffic volume transferred is 4 Kbytes, or if it is used by the end of the connection
    - The script *packet-segment-logging.bro* enables logging of packet data in spite of protocol parsing violation being detected
    - These data can then be used by other analyzers invoked by the Event module

Policy script interpreting

- The Bro script set structure (58)
  - Metrics
    - Contains scripts that make various statistics about connections in general, HTTP requests and responses, and SSL connections
    - These scripts should be considered examples and as such they should be further developed in order to be used in practice

Policy script interpreting

- The Bro script set structure (59)
  - Signatures
    - This framework is currently capable of detecting so-called *reverse shells*
      - Attacks attempting to execute system commands on the compromised host

Policy script interpreting

- The Bro script set structure (60)
  - Software
    - Provides support for defining vulnerable versions of software
    - A notice will be generated if such a version (or an older one) is detected
    - In addition, these scripts are capable of defining software names for which it is possible to track version changes
    - In such a case, a notice is generated if the software version changes on a specified host

Policy script interpreting

- The Bro script set structure (61)
  - The Integration script subset
    - Provides support for gathering information about attacks from various systems
    - Currently, it supports Snort alert integration into Bro
    - To this end, the support for Barnyard2 is provided
    - This enables aggregation of data from Snort and Bro in a database

Policy script interpreting

- The Bro script set structure (62)
  - The Misc script subset
    - Contains miscellaneous scripts, for example a script that turns on profiling of Bro resource consumption, memory and packet statistics, manages trace files, etc.
    - A script that detects port scans is often included in this category

Policy script interpreting

- The Bro script set structure (63)
  - The Protocols script subset
    - Provides additional support for detecting events related with various protocols
    - Currently, additional support is provided for the following protocols: DNS, FTP, HTTP, SMTP, SSH, and SSL
    - In addition, there is a special script set CONN that extracts relevant information about hosts and services known to the defended network, as well as information about weird traffic

Policy script interpreting

- The Bro script set structure (64)
  - CONN
    - Provides support for logging information about hosts that have performed complete TCP handshakes during the day, as well as information about services used in the defended network
    - In addition, classification of so-called *weird* events is possible in order to further analyze these events

Policy script interpreting

- The Bro script set structure (65)
  - DNS
    - Contains scripts capable of analyzing situations in which external names resolve to addresses within the defended network
    - Also defines additional responses to a query to the DNS log

Policy script interpreting

- The Bro script set structure (66)
  - FTP
    - Provides support for detecting potentially malicious activities related with the FTP protocol
    - Also capable of detecting client and server software used with the FTP protocol

Policy script interpreting

- The Bro script set structure (67)
  - HTTP
    - Provides additional support for detection of software and Web applications used with the HTTP protocol, browser plugins, extraction of cookies, uri identifiers and header names
    - Also capable of detecting malware with known hash values as well as SQL injection attacks

Policy script interpreting

- The Bro script set structure (68)
  - SMTP
    - Provides support for detecting blocked senders and recipients, suspicious places of origin and E-mail related software, for example E-mail client software and Web mail software

Policy script interpreting

- The Bro script set structure (69)
  - SSH
    - Provides heuristics to determine whether the brute force password guessing attack has taken place
    - Also capable of detecting software used as client and server for SSH, as well as SSH connections originating from suspicious hosts

Policy script interpreting

- The Bro script set structure (70)
  - SSL
    - Provides support for certificate extraction, validation and logging
    - Logs expired certificates
    - Also contains an auxiliary script that performs computation of MD5 hash values for certificates

Policy script interpreting

- The Bro script set structure (71)
  - The Tuning script subset
    - Contains various scripts that have influence on the amount of resources used by Bro
    - Contains a script that defines which elements of the defended network will be tracked by Bro
      - By default, hosts, services and certificates will be tracked
    - Also contains a script that captures TCP fragments
    - Another script issues a warning if Bro is not configured properly, for example if so-called *local network* is not defined

Policy script interpreting

- The Bro script set structure (72)
  - The Site script set (1)
    - Contains a script named *local.bro*
      - Defines the local site policy
      - The operator is supposed to edit this script as appropriate
      - Defines logging of loaded scripts at the execution time, Bro tuning settings, local networks, policy related with software detected by Bro, the protocols from the Policy subset that will be loaded, etc.
    - The Site script group also defines local details of Bro cluster, if it is used

Policy script interpreting

- The Bro script set structure (73)
  - The Site script set (2)
    - Demo: Display the contents of the file *local.bro*

      (location: /usr/local/bro/share/bro/site/local.bro)

Policy script interpreting

- Messages to the operator (1)
  - Bro produces a lot of information about the analyzed network
  - In addition to a very detailed information about connections contained in various logs, Bro produces a number of messages to the operator called *notices*
    - These are contained in separate log files, *notice.log* and *weird.log*

Policy script interpreting

- Messages to the operator (2)
  - A notice is a message from Bro that reports suspicious behavior of certain traffic elements
  - Most often notices escalate into alerts, but it is possible to just log them or ignore them
  - A notice contains information about suspicious behavior, such as the notice type (which must be defined in advance in the corresponding policy script), source information, etc.

Policy script interpreting

- Messages to the operator (3)
  - A special kind of notice is a *WeirdActivity* notice
    - Reports activity that does not conform to standard protocols
    - This might be result of errors in protocol implementation, errors in communication, but this might also be a consequence of attack activity (e.g. IDS evasion attacks)
    - Thus, the user can also escalate the WeirdActivity messages into alarms, but most often they are just logged

Policy script interpreting

- Messages to the operator (4)
  - The goal of separation of notices into real notices and weird activity notices is reduction of the number of false positives
  - Obviously, real notices are only generated if the corresponding policy scripts interpret the events discovered by the Event module as suspicious
  - These scripts then must define the notice types and other information, which is to be contained in the notices

Policy script interpreting

- Messages to the operator (5)
  - The weird activity notices are not generated by the corresponding policy scripts
  - Instead, they are generated by the protocol analyzers if they detect inconsistencies in protocol implementations or some other problems with the analyzed protocols

Policy script interpreting

- Messages to the operator (6)
  - Example – a real notice (1)

    1336051829.145903 ------SSH::Password_Guessing Threshold crossed by metric_index(host=128.39.140.105) 30/30 - 128.39.140.105 --30 bro Notice::ACTION_LOG, Notice::ACTION_ALARM 6,5 3600.000000 F ----- 128.39.140.105 --

Policy script interpreting

- Messages to the operator (7)
  - Example – a real notice (2)
    - This notice warns about an SSH password guessing activity from the host with the IP address 128.39.140.105
    - By default, the number of unsuccessful password trials for SSH is set to 30 in Bro and the notice reports that the threshold was exceeded
    - This notice will be logged (ACTION_LOG) and an alert will be raised (ACTION_ALARM)
    - The first number in this notice is the time stamp

Policy script interpreting

- Messages to the operator (8)
  - Example – a weird activity notice (1)

    1336051838.706844 xNprNdVSQ88 128.39.140.91 36391 128.39.140.220 80 data_before_established -F bro

Policy script interpreting

- Messages to the operator (9)
  - Example – a weird activity notice (2)
    - This weird activity notice reports about data transfer from the host with the IP address 128.39.140.91, port 36391, to the host with the IP address 128.39.140.220, port 80, before the TCP connection was established
    - Possible reason for this is portscan activity, but it is up to the operator to investigate this further and eventually escalate such events into alerts

Policy script interpreting

- Policy scripts – Examples (1)
  - Example – escalating a notice to an alert (1)
    - Portscanning activity observed from a particular group of hosts

```
const suspicious_scanners : set[addr] = {
        128.39.140.170,
        128.39.140.171,
} &redef;
```

Policy script interpreting

- Policy scripts – Examples (2)
  - Example – escalating a notice to an alert (2)

```
redef Notice::policy += {
[$action = Notice::ACTION_ALARM,
$pred(n: Notice::Info) ={
        return n$note == Scan::PortScanSummary &&
        n$src in suspicious_scanners;
        }
]
};
```

Policy script interpreting

- Policy scripts – Examples (3)
  - Example – escalating a notice to an alert (3)
    - The first part of the script defines a set of suspicious IP addresses
      - Even though the set is constant, it is possible to change the values in the set, for example in another script or later in the same script, since the *redef* keyword is present
    - The redef keyword is also used in the second part of the script, this time to indicate a change in an already existing constant set
      - We change the constant set *policy* defined in the Notice framework main script *main.bro*

Policy script interpreting

- Policy scripts – Examples (4)
  - Example – escalating a notice to an alert (4)
    - The notice policy is defined as a constant set of policy items
    - Each policy item is defined between square brackets
    - Since the redef keyword is present, it is possible to modify the default policy set
    - This is done in the second part of our script
    - We add a new policy item in the notice policy

Policy script interpreting

- Policy scripts – Examples (5)
  - Example – escalating a notice to an alert (5)
    - The action of the new policy item is to raise an alert (ACTION_ALARM)
    - *pred* is a Boolean function that returns the logical value of the expression given in the return statement
      - In our case, it will be *true* when the notice *Info* record element *note* is equal to Scan::PortScanSummary and the notice Info record element *src* has a value equal to some element of the suspicious scanners set
    - The Info record and the policy item are also defined in the main script of the Notice framework *main.bro*

Policy script interpreting

- Policy scripts – Examples (6)
  - Example – escalating a notice to an alert (6)
    - The $ sign is the dereference operator
      - Addresses record fields in the Bro scripting language
    - The notice Scan::PortScanSummary is generated by the Bro script *scan.bro*
    - In the default notice policy it can be observed that only the notice types contained in the set *alarmed types* will cause raising alerts
    - By default, this set is empty

Policy script interpreting

- Policy scripts – Examples (7)
  - Example – escalating a notice to an alert (7)
    - If we want to make Bro raise alerts on notices of the type PortScanSummary, we have to include that notice in the set of alarmed types
    - It is a constant set with the redef keyword, so we can add an element into it by the following script fragment

```
redef Notice::alarmed_types += {
   Scan::PortScanSummary
};
```

Policy script interpreting

- Policy scripts – Examples (8)
  - Example – escalating a notice to an alert (8)
    - However, this statement will cause Bro raising alerts on portscans originating from any host, not only those from our set of suspicious scanners
    - Because of that, the correct way to alert only on portscans originating from the set of suspicious scanners is to modify the policy, as we have done in our example

Policy script interpreting

- Policy scripts – Examples (9)
  - Example – escalating a notice to an alert (9)
    - The script *scan.bro* is considered to exist in advance
    - The Bro system must be instructed to load it in order to be used
    - This means that it must be placed in one of the standard Bro directories (for example /policy/misc) and then the site policy script *local.bro* must be updated by including the *@load* statement (for example, @load misc/scan.bro)

Policy script interpreting

- Policy scripts – Examples (10)
  - Example – escalating a notice to an alert (10)
    - Demo
      - Display the contents of the file *main.bro* of the notice framework
      - Run the Example 1
        - » The script *scan.bro* is here:
          - » http://git.bro-ids.org/bro-scripts.git/tree
        - » The instructions for running the example are in the file Example1.txt

Policy script interpreting

- Policy scripts – Examples (11)
  - Example – defining a new notice from an event (1)
    - Suppose we would like to raise an alert whenever an HTTP request is sent from a suspicious client to the defended Web server in the local network

```
const suspicious_clients : set[addr] = {
   128.39.140.60,
   128.39.140.61,
   128.39.140.62,
} &redef;
```

Policy script interpreting

- Policy scripts – Examples (12)
  - Example – defining a new notice from an event (2)

```
export{
        redef enum Notice::Type +={
        HTTP::SuspiciousRequest
        };
}
```

Policy script interpreting

- Policy scripts – Examples (13)
  - Example – defining a new notice from an event (3)

```
redef Notice::policy += {
[$action = Notice::ACTION_ALARM,
$pred(n: Notice::Info)={
        return n$note == HTTP::SuspiciousRequest;
        }
]
};
```

Policy script interpreting

- Policy scripts – Examples (14)
  - Example – defining a new notice from an event (4)

```
event http_request(c: connection, method: string,
original_URI: string, unescaped_URI: string,
version: string){
        if(c$id$orig_h in suspicious_clients){
        NOTICE([$note=HTTP::SuspiciousRequest,
        $src=c$id$orig_h,
        $msg="HTTP request from a suspicious client"]);
        };}
```

Policy script interpreting

- Policy scripts – Examples (15)
  - Example – defining a new notice from an event (5)
    - In the first part of the script, we define a set of suspicious IP addresses
    - In the second part of the script, we declare a new notice type *SuspiciousRequest*
    - The *export* statement instructs Bro to include the new notice in the global notice set in the category of HTTP-related notices
      - Without that statement, we could not relate the new notice with HTTP events

Policy script interpreting

- Policy scripts – Examples (16)
  - Example – defining a new notice from an event (6)
    - The next part of the script instructs Bro to raise an alert on the notice, i.e. it escalates the notice into an alert
    - The final part of the script defines the event handler that generates the notice

Policy script interpreting

- Policy scripts – Examples (17)
  - Example – defining a new notice from an event (7)
    - In Bro, many events generated by the event module already have determined event types
    - The list of the existing event types is available in the file *events.bif.bro*, which is located in the *base* set of scripts
    - For each event type, a set of corresponding parameters is defined, which are passed to the event handler that is written by the user

Policy script interpreting

- Policy scripts – Examples (18)
  - Example – defining a new notice from an event (8)
    - In this particular case, the event generated by the event module is http_request and the parameters passed to the event handler are the connection record, the method used in the http request (for example GET), the original URI, the unescaped URI (i.e. cannonicalized URI, without UNICODE characters and escape sequences), and the version of the HTTP protocol used in the request

Policy script interpreting

- Policy scripts – Examples (19)
  - Example – defining a new notice from an event (9)
    - To define the notice, we are going to analyze the connection record *c* passed to the event handler
    - The connection record contains several fields, some of which are also records
      - For example, it contains the *id* field, which is a record containing the IP address of the sender (orig_h), the port of the sender (orig_p), the IP address of the responder (resp_h), and the port of the responder (resp_p).

Policy script interpreting

- Policy scripts – Examples (20)
  - Example – defining new notice from an event (10)
    - The connection record is defined in the script *init-bare.bro*, located in the *base* set of scripts
    - We get the IP address of the sender by dereferencing: cidorig_h
    - In the if statement used in the event handler we check whether that IP address belongs to the already defined set of suspicious IP addresses

Policy script interpreting

- Policy scripts – Examples (21)
  - Example – defining new notice from an event (11)
    - In the case it belongs to that set, we raise the notice
    - This is a new notice, so we call the NOTICE function and provide the set of notice parameters in the form of a record
    - In this case we use the *note* field and set its value to the value already declared in the beginning of the script
    - Then we use the *src* field to log the originating IP address of the suspicious traffic

Policy script interpreting

- Policy scripts – Examples (22)
  - Example – defining new notice from an event (12)
    - Finally, we define the message to be logged by setting the value of the *msg* field to *HTTP request from a suspicious client*
    - Demo: Run the Example 2
      - The instructions for running the example are in the file Example2.txt

Policy script interpreting

- Policy scripts – Examples (23)
  - Example – anomaly detection with Bro (1)
    - Suppose we want to alert on http requests coming from clients not listed in the set of allowed clients
    - This is anomaly detection, as we define what is normal and all deviations from normal are reported

Policy script interpreting

- Policy scripts – Examples (24)
  - Example – anomaly detection with Bro (2)

```
const allowed_clients : set[addr] = {
        128.39.140.60,
        128.39.140.61,
        128.39.140.62,
} &redef;
```

Policy script interpreting

- Policy scripts – Examples (25)
  - Example – anomaly detection with Bro (3)

```
export{
        redef enum Notice::Type +={
        HTTP::SuspiciousRequest
        };
}
```

Policy script interpreting

- Policy scripts – Examples (26)
  - Example – anomaly detection with Bro (4)

```
redef Notice::policy += {
[$action = Notice::ACTION_ALARM,
$pred(n: Notice::Info)={
        return n$note == HTTP::SuspiciousRequest;
}
]
};
```

Policy script interpreting

- Policy scripts – Examples (27)
  - Example – anomaly detection with Bro (5)

```
event http_request(c: connection, method: string,
original_URI: string, unescaped_URI: string,
version: string){
        if(!(c$id$orig_h in allowed_clients)){
        NOTICE([$note=HTTP::SuspiciousRequest,
        $src=c$id$orig_h,
        $msg="HTTP request from a suspicious client"]);
        };}
```

Policy script interpreting

- Policy scripts – Examples (28)
  - Example – anomaly detection with Bro (6)
    - The code is very similar to that presented in the previous example
    - Only the event handler for processing http request events is different in the sense that the notice is now raised if the request *does not* originate from the set of allowed addresses
    - Demo: Run the example 3
      - The instructions for running the example are in the file Example3.txt

Policy script interpreting

- Logs and alerts in Bro (1)
  - Bro performs intensive logging of the activity in the defended system, but this logging is indirect, namely the activity is encoded by the corresponding events
  - At the lowest level of logging, every connection is interpreted as an event and logged as such

Policy script interpreting

- Logs and alerts in Bro (2)
  - Example - a record in the connection log file *conn.log* (1)

    1336051819.343513 E6jSfCJ7BNj 128.39.140.91 63392

    128.39.140.220 2909 tcp ----S0 T 0 S 1 44 0 0

    - This record gives the operator the following information (1)
      - Time stamp
      - The originating IP address and port
      - The receiving IP address and port

Policy script interpreting

- Logs and alerts in Bro (3)
  - Example - a record in the connection log file *conn.log* (2)
    - This record gives the operator the following information (2)
      - The protocol
      - Whether the connection was successfully established or not (in this particular case, the code S0 means that the connection attempt was observed, but there was no reply)
      - The number of bytes present in the originating IP traffic (in this case, 44 bytes)

Policy script interpreting

- Logs and alerts in Bro (4)
  - Example - a record in the connection log file *conn.log* (3)
    - The header of the log file conn.log contains information about the meaning of the record fields
    - Each record (i.e. connection) is assigned a unique identification code (in this particular case, the code is E6jSfCJ7BNj)

Policy script interpreting

- Logs and alerts in Bro (5)
  - In general, the events generated in the event module determine also which log files will be created
  - The *conn.log* is present if any connection is established during the activity of Bro
  - The *dns.log* reports events related with Domain Name Service (DNS) and is normally present

Policy script interpreting

- Logs and alerts in Bro (6)
  - The log file *weird.log* is very often present, since there are many inconsistencies in Internet traffic and the Bro event module is not perfect in determining which protocol is used in the traffic at hand

Policy script interpreting

- Logs and alerts in Bro (7)
  - The *notice.log* file is present if any notices were generated during the execution of Bro
  - By default, all the notices are just logged and they appear in this file
  - If we have escalated some notices to alerts, a new file *notice_alarm.log* is created, where such notices are logged, in addition to logging to the file notice.log

Policy script interpreting

- Logs and alerts in Bro (8)
  - The log file *notice_policy.log* is always present
  - It contains information about actions that will be triggered by various events
  - If any event related with protocols supported by Bro were generated while Bro was active, the corresponding log file is created
    - Example
      - The log file *http.log* is created if http-related events were generated by Bro

Policy script interpreting

- Logs and alerts in Bro (9)
  - In addition to the log files related with the analyzed traffic, there are several log files that store information about the particular Bro session
    - The log file *loaded_scripts.log* contains the list of all the scripts loaded in the particular Bro session
    - The log file *reporter.log* stores information about the execution of Bro
      - Among other elements, the number of dropped packets is reported in that file

Bro signatures

- In addition to its own original approach to intrusion detection, Bro also offers classical signature-based intrusion detection
  - Bro uses its own signature language
    - Different from Snort
    - Offers much more possibilities than Snort
    - Not as easy to understand as Snort signatures
    - Not compatible with Snort signatures
      - The snort2bro conversion script no longer maintained

Bro signatures

- Bro signature language (1)
  - Uses regular expressions
  - Example (1)

```
signature sig1 {
        ip-proto == tcp
        dst-port == 80
        payload /.*asp/
        event “Suspicious traffic!" }
```

Bro signatures

- Bro signature language (2)
  - Example (2)
    - This signature matches the regular expression .*asp on the TCP connections directed towards the port 80
    - If a match is found, the signature_match event is raised

      event signature_match(state: signature_state, msg: string, data: string)

Bro signatures

- Bro signature language (3)
  - Example (3)
    - In the event signature_match
      - signature_state contains information about the connection that triggered the signature
      - msg is the string specified by the event statement in the signature (Suspicious traffic!)
      - data is the last part of payload that triggered the signature pattern match

Bro signatures

- Bro signature language (4)
  - To escalate a signature match into an alert, we use the *main.bro* script of the *signature* framework from the *base* script category
    - A default event handler in that script raises, among other notices, the notice Signatures::Sensitive_Signature

Bro signatures

- Bro signature language (5)
  - We specify the file name where the signatures are by extending the variable signature_files in the script init-bare.bro by means of the += operator
  - Demo: run example 4
    - bro –i em1 local policy1.bro
    - Display the signature file *sig1.sig*
    - Display the bro log files after the http request

Bro signatures

- Bro signature language (6)
  - A signature has the format

    signature <id> { <attributes> }
  - <id> is a unique label for the signature
  - Two types of attributes
    - Conditions
    - Actions
  - Conditions define when the signature matches
  - Actions declare what to do in the case of a match

Bro signatures

- Bro signature language (7)
  - Conditions are divided into 4 categories
    - Header
    - Content
    - Dependency
    - Context

Bro signatures

- Bro signature language (8)
  - Header conditions (1)
    - Limit the applicability of the signature to a part of traffic that contains matching packet headers
      - For TCP, this match is performed only for the first packet of a connection
      - For other protocols, it is done on each individual packet

Bro signatures

- Bro signature language (9)
  - Header conditions (2)
    - In the Header condition, we define the source and destination IP addresses, the protocol, and the source and destination ports

Bro signatures

- Bro signature language (10)
  - Content conditions (1)
    - Defined by regular expressions
    - Two kinds of content conditions
      - Declared with the *payload* statement
        - » Matched against the raw payload of a connection (for reassembled TCP streams) or that of each packet (for ICMP, UDP, and non-reassembled TCP)
      - Prefixed with an analyzer-specific label
        - » Matched against the data extracted by the corresponding analyzer

Bro signatures

- Bro signature language (11)
  - Content conditions (2)
    - A payload condition is of the form

      payload /<regular expression>/
    - An analyzer-specific content condition is of the form

      http-request /<regular expression>/

      where instead of http-request other keywords may appear (http-request-header, http-request-body, http-reply-header, http-reply-body, ftp, etc.)

Bro signatures

- Bro signature language (12)
  - Dependency conditions (1)
    - Define dependencies between signatures
    - Two conditions (1)

      requires-signature [!] <id>

      » Current signature matches only if the signature given by id matches for the same connection

      » Using ! negates the condition

Bro signatures

- Bro signature language (13)
  - Dependency conditions (2)
    - Two conditions (2)

      requires-reverse-signature [!] <id>

      » Similar to requires-signature, but id has to match for the opposite direction of the same connection, compared to the current signature

      » This allows to model the notion of requests and replies

Bro signatures

- Bro signature language (14)
  - Context conditions (1)
    - Pass the match decision on to other components of Bro
    - Only evaluated if all other conditions have already matched
    - The context conditions
      - eval <policy-function>
      - payload-size <cmp> <integer>
      - same-ip
      - tcp-state <state-list>

Bro signatures

- Bro signature language (15)
  - Context conditions (2)
    - eval <policy-function>
      - The given policy function is called and has to return a Boolean value confirming the match
      - If false is returned, no signature match is going to be triggered
    - payload-size <cmp> <integer>
      - Compares the integer to the size of the payload of a packet
      - For reassembled TCP streams, the integer is compared to the size of the first in-order payload portion

Bro signatures

- Bro signature language (16)
  - Context conditions (3)
    - same-ip
      - Evaluates to *true* if the source address of the IP packets equals its destination address

Bro signatures

- Bro signature language (17)
  - Context conditions (4)
    - tcp-state <state-list>
      - Imposes restrictions on the current TCP state of the connection
      - state-list is a comma-separated list of the keywords
        - » established (the three-way handshake has already been performed)
        - » originator (the current data is send by the originator of the connection)
        - » responder (the current data is send by the responder of the connection)

Bro signatures

- Bro signature language (18)
  - Actions (1)
    - Define what to do if a signature matches
    - Two possible actions

      event <string>

      enable <string>

Bro signatures

- Bro signature language (19)
  - Actions (2)
    - event <string>
      - Raises a signature_match event
      - The event handler

        event signature_match(state: signature_state, msg: string, data: string)

        » The given string is passed in as msg

        » data is the current part of the payload that has eventually lead to the signature match

Bro signatures

- Bro signature language (20)
  - Actions (3)
    - enable <string>
      - Enables the protocol analyzer <string> for the matching connection ("http", "ftp", etc.)
      - Used by Bro Dynamic Protocol Detection (DPD) to activate analyzers

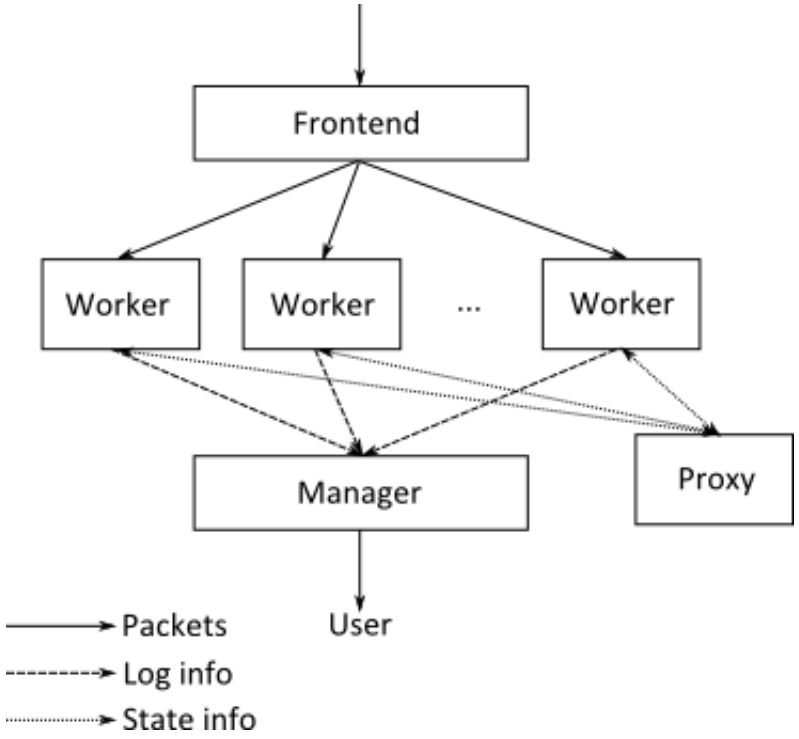
Bro cluster

- Bro is not multithreaded
  - Once the limitations of a single processor core are reached, Bro cluster spreads the workload across many cores/physical computers

Bro cluster

- The Bro cluster framework tools and scripts perform management of many Bro processes
  - Examine packets and perform correlation activities
  - Act as a single entity
- The components of a Bro cluster
  - Frontend
  - Manager
  - Workers

Bro cluster


Frontend
Worker
Worker
...
Worker
Manager
Proxy
User
Packets
Log info
State info

Bro cluster

- Frontend
  - A load balancer
  - Often a hardware device
- Manager
  - A Bro process
  - Two primary tasks
    - Receiving log messages and notices from the rest of the nodes in the cluster using the Bro communications protocol
    - De-duplicating notices

Bro cluster

- Worker
  - A Bro process
  - Intercepts network traffic and performs protocol analysis on the reassembled traffic streams
  - Most of the work of an active cluster takes place on the workers
  - The workers typically represent the majority of the Bro processes that are running in a cluster

Bro cluster

- Proxies
  - Special Bro processes
  - In addition to the components of a Bro cluster, the proxies synchronize the states of the workers, thus making connections of the workers with other workers unnecessary
  - By means of proxies, the workers share information about the known hosts and services etc.