CIDSS: Common Intrusion Detection Signatures Standard

Adam Wierzbicki, Jacek Kaliński and Tomasz Kruszona
Polish-Japanese Institute of Information Technology

Introduction

Intrusion Detection Systems (IDS) are controlled by decision rules (signatures), which contains technical description of intrusion accident and action that must be taken when this intrusion occur (e.g. send an email, write to system log).
Nowadays, there are a lot of IDS: commercial as well as open source.
Choosing the suitable IDS, configuration and management is a very hard and responsible task because each of well known system is using theirs own signatures standard.

In 2003 year process of creating standard has been launched to exchange information between different IDS. An Internet Engineering Task Force (IETF) working group (IDWG, Intrusion Detection Working Group) standardized that mechanism.
But still there was no tries to create a standard of IDS signatures. Because there is no common standard, administrators who know one system must learn other IDS from the beginning. This is common situation, because administrators usually start with the Snort (most popular open source IDS). When they get a job in a company, which have a commercial IDS, they just have to learn a new one. Similar situations we can find when we change a job or when we are implementing a new Intrusion Detection System in our company.
Also there is a problem when independent institutions would like to do an audit of IDS. Because signatures are not standardized an audit becomes very hard or even it is impossible to realize. In this kind of audit it is necessary to analyze signatures. But lets try to compare signatures when you don’t know a format of signatures of different IDS’s. The same difficulty is when a company will try to integrate different (maybe inherited?) Intrusion Detections Systems in common security policy.

Common Intrusion Detection System Signatures

Meanwhile, a main part of signature (especially systems specializing in network data analysis) is similar in various systems. We can try to translate signatures automatically, but we have already checked that some IDS signatures cannot be translated from one system to another. No standard in this category cause signatures to be translated between themselves (different IDS).
Additionally, translation is hard because in most cases there is no detailed signatures specification.

In 2004 we have appeared proposal to create a standard in signatures format – Common Intrusion Detection Signatures Standard (CIDSS) submitted to IETF [1] as an Internet Draft. Currently it is third (draft-wierzbicki-cidss-02) version of an Internet Draft. The process of standardization can allow improving standard and creating tools for automatic signatures translating, especially with cooperation with IDS authors. CIDSS is based on Extensible Markup Language (XML), so it is ready to automatically parse, verify and extend signatures. An XML signatures are human-readable and easier to analyze than rules of other IDSs.

Our standard is based on experiences when we were trying to translate signatures automatically between various systems and is a "sum" of possibilities expressing information about intrusion accidents. CIDSS could be an introduction to write tools for automatic translations [2]. SigTranslator has been written in parallel with standard and is currently supporting IDS: Snort, Dragon, Shoki and RealSecure with partial translating support. Full translation some of signatures to other system (e.g. Snort to Dragon) is impossible (due to limitations in Dragon signatures), but all signatures could be translated to CIDSS.

Because we want to integrate various signature standards, CIDSS is now the most extensible language that can describe network intrusions. Supporting Snort signatures make CIDSS to implement features such as stateful rules. In effect it is necessary to make CIDSS more flexible and extensible.

Example of CIDSS signature

As an example of CIDSS signature, lets analyze the following situation: we got information that from our network there was an attempt of communication between client and TFN (Tribe Flood Network) daemons. Client is installed inside our network and it tries to communicate with hosts from `83.0.0.0/8` and `84.0.0.0/8` networks and with host `194.154.2.142`. We would like to know which host from our network is a TFN client. A client can communicate with other daemons to start remote attack. Daemons on compromised hosts respond on client requests for open shell with ICMP response no.123, sequence number 0 and string “`shell bound to port`”. IDS should send alert to administrator (or generate appropriate log entry).

Here is a sample CIDSS rule creation process.
Most external tag `Signatures` is opening the rule set. Each rule resides in `Signature` mark. In every `Signatures` tag there could be one or more `Signature` tags. In conclusion CIDSS document contains one or more independent rules. Template can look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<Signatures xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="common.xsd">
  <Signature sid="RULE_NUMBER">
    <Enabled>true</Enabled>
  </Signature>
</Signatures>
```

Diagram1 Signature and Signatures tags.

`Enabled` tag defines which of rules included in `Signatures` should be considered active by IDS (which signature is enabled).

In each rule we must define used protocol and characteristics of a protocol. Protocol tags depends on protocol type as described in Internet Draft [1]. E.g. for ICMP we can define ICMP packets type. In this sample, value of `ICMP_Itype` is 0, so rule matches all ICMP types.

```
<Signatures xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="common.xsd">
  <Signature sid="RULE_NUMBER">
     <Enabled>true</Enabled>
     <Protocols>
       <Protocol Type="icmp">
         <ICMP_Icmp_Id>123</ICMP_Icmp_Id>
         <ICMP_Icmp_Seq>0</ICMP_Icmp_Seq>
         <ICMP_Itype>0</ICMP_Itype>
       </Protocol>
     </Protocols>
  </Signature>
</Signatures>
```

Diagram2 Protocol information

As shown above, `Protocols` tag contains one `Protocol` tag, but if rule applies to more than one protocol then it can contain more `Protocol` tags – e.g. information for UDP or TCP protocol. Then `Protocols` tag could looks like:

```
     <Protocols>
       <Protocol Type="tcp">
         <TCP_Ack>0</TCP_Ack>
         <TCP_Window>34000</TCP_Window>
       </Protocol>
       <Protocol Type="udp">
         <UDP_Dsize>40000</UDP_Dsize>
       </Protocol>
     </Protocols>
```

Diagram3 How to use multiple protocol tags

It is similar when we define source (`Source`) and destination (`Destination`) of potential attack – it is possible to describe more then on source and destination in each rule. In our example rule we know there could be any destination (any IP address – TFN client) and source are hosts with prefix `83` or `84` or host with `194.154.2.142` address (TFN daemons).

```
<Signatures xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="common.xsd">
  <Signature sid="NR_REGULY">
    <Enabled>true</Enabled>
    <Protocols>...</Protocols>
    <Sources>
      <Source Src_ID="1">
       <Source_IP Neg="false" Mask="8">83.0.0.0</Source_IP>
      </Source>
      <Source Src_ID="2">
       <Source_IP Neg="false" Mask="8">84.0.0.0</Source_IP>
      </Source>
      <Source Src_ID="3">
       <Source_IP Neg="false" Mask="32">194.154.2.142</Source_IP>
      </Source>
      <Src_Logic>1 OR 2 OR 3</Src_Logic>
    </Sources>
    <Destinations>
      <Destination Dst_ID="1">
          <Destination_IP>any</Destination_IP>
      </Destination>
    </Destinations>
  </Signature>
</Signatures>
```

Diagram 4 Source and destination description

Lets we have a look at `Src_Logic` tag. When there is more then one `Source` tag, we can use logical expression to define dependencies between sources. We used in our example an alternative because we want to match any of sources. There can be used even complex logical expression with one of the keywords: `AND`, `OR`, `NOT` and brackets. We can handle `Destinations`, `Protocols`, `Patterns` tags the same way as `Sources`. `Patterns` tag is one of the most important tags in whole signature. It contains `Pattern` tags which are describing packets using particular pattern. For now, supported pattern types are: PCRE expression, string, decimal and hexadecimal value. This is how our rule looks like with use of `Patterns`:

```
<Signatures xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="common.xsd">
  <Signature sid="NR_REGULY">
    <Enabled>true</Enabled>
    <Protocols>...</Protocols>
    <Sources>...</Sources>
    <Destinations>...</Destinations>
    <Patterns id="1">
      <Pattern pat_id="1">
        <Pattern_Type>string</Pattern_Type>
        <Pattern_Content CaseSensitive="true">shell bound to \
        port</Pattern_Content>
        <Pattern_Offset>36</Pattern_Offset>
      </Pattern>
    </Patterns>
  </Signature>
</Signatures>
```

Diagram5 Patterns content in a signature

Look, we can define at which offset from the end of packet header, we should look for specific pattern. This is possible using a `Pattern_Offset` tag.

Last we can define an action of rule and describe whole signature:

```
<Signatures xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="common.xsd">
  <Signature sid="NR_REGULY">
    <Enabled>true</Enabled>
    <Sig_source>Snort</Sig_source>
    <Description>DDOS TFN server response</Description>
    <Action>alert</Action>
    <Protocols>...</Protocols>
    <Sources>...</Sources>
    <Destinations>...</Destinations>
    <Patterns id="1">...</Patterns>
    <Message>DDOS TFN server response</Message>
    <Comment>reference:arachnids,182</Comment>
    <Comment>classtype:attempted-dos</Comment>
    <Comment>rev:6</Comment>
  </Signature>
</Signatures>
```

Diagram6 Message, action, source of rule and rule description

Using `Messsage`, `Comment` or `Description` tags we can easily and precisely describe our rule. `Sig_Source` contains information about source of translation for the particular signature.

Rule analyzed above has been created as a result of translation from Snort IDS. Next it was translated to Shoki and Dragon rules:

Snort:

```
alert icmp $HOME_NET any -> $EXTERNAL_NET any (msg:"DDOS TFN
server response"; icmp_id:123; icmp_seq:0; itype:0;
content:"shell bound to port"; reference:arachnids,182;
classtype:attempted-dos; sid:238; rev:6;)
```

Dragon:

```
I D F B 10 0 0 IDS182:ddos_ddos-tfn-server-response
shell/2f20bound/2f20to/2f20port
```

Shoki:

```
icmp and (icmp[4:2] == 123) 65536 SEARCH IDS182 ddos_ddos-tfn-
server-response 'shell bound to port' ALL 1 NULL
```

Diagram7 Equivalent rules for Snort, Shoki and Dragon

As you can see, rules used in IDS are much shorter than CIDSS. But it is the only one advantage. They are not human-readable, and we can make a mistake during creating or modifying signatures. On shown example we can see problem with manual translations from various IDS: this rules are not comparable, but contains the same information! For example, we want to do an external audit. If people who do this audit know well Snort system, he will probably have troubles with Dragon or Shoki IDS. Even we have experts which know all of this systems, probability to make a mistake is increasing significantly during comparison IDS rules used in a company with schemas written for other systems. Doing an audit will be possible only when have sample rules for different IDS.

Signatures and sessions

Currently the only one IDS that support stateful rules is Snort. Under the term of stateful sessions we understand that packets processed by system are considered in context of packets previously processed. Snort interpretation of stateful rules is limited to protocols of transport layer (UDP or TCP). Using session we are able to describe more complex types of attacks. Such attacks like port scanning can be invisible for IDS that doesn't take session state in consideration. Snort will have problems with detection of slow port scanning from different sources. It cannot analyze packet flow that doesn't start from transport protocol (e.g. malicious routing modification attempts).
In Snort multiple rules can be related through session state. Rules are related by operations made on particular session variables (checking and modifying a value of variables)
CIDSS is able to define signatures containing sessions in wide meaning – it considers state of packet flows of any protocol. So, it is also possible to translate Snort stateful signature to CIDSS signature.
On diagram8 we can see example rule containing `Session` tag and considers session state. On this example we would like to explain how tags describe above work. Lets imagine that attacker is trying to make a DoS attempt using Microsoft RPC service. He is sending multiple requests to make system vulnerable. It takes all system resources. System after detecting attack attempt should correctly react.
`Source_IP` and `Destination_IP` tags contains `$EXTERNAL_NET` and `$HOME_NET` values that are specific values defined in IDS. IDS have possibility to define in configuration files or in system environment such values, so CIDSS is capable to use them too. In our example we use also `Pattern` and `Pat_Logic` tags.
We will explain meaning of these tags in example from diagram8. First tag in `Session` section is `Session_End`, which describes conditions for ending session. Session ends after 60 seconds as `Session_Timeout` defines, or after 20 packets from start of session. After session end a packet counter is zeroed so in 60 seconds 20 packets must match the rule (conditions defined in signature). Moreover, packets in session must match additional conditions defined by `Session_Case` (contained in `Session_Instructions`). First tag (`Direction`) defines direction of packets counted into session – from source (host starting session) to destination. Condition described in `Case_State_Condition` applies to session variables. In this case variable dce.isystemactivator.bind.call.attempt must be set by another signature.
To modify variable values we can use `Case_State_Instructions` tag.

```
<Signature sid="2494">
   <Enabled>true</Enabled>
   <Sig_source>Snort</Sig_source>
   <Description>NETBIOS DCEPRC ORPCThis request flood
attempt</Description>
   <Action>alert</Action>
   <Protocols><Protocol Type="tcp">
       <TCP_State>established</TCP_State>
   </Protocol> </Protocols>
   <Sources><Source src_id="1">
      <Source_IP>$EXTERNAL_NET</Source_IP>
      <Source_Port>any</Source_Port>
   </Source></Sources>
   <Destinations><Destination dst_id="1">
      <Destination_IP>$HOME_NET</Destination_IP>
      <Destination_Port>135</Destination_Port>
   </Destination></Destinations>
   <Patterns id="1">
      <Pattern pat_id="1">
       <Pattern_Type>hex</Pattern_Type>
       <Pattern_Content>05</Pattern_Content>
       <Pattern_Within>1</Pattern_Within>
      </Pattern>
      <Pattern pat_id="2">
       <Pattern_Type>hex</Pattern_Type>
       <Pattern_Content>00</Pattern_Content>
       <Pattern_Within>1</Pattern_Within>
       <Pattern_Distance>1</Pattern_Distance>
   <Pattern_Byte_test><![CDATA[1,&,1,0,relative]]></Pattern_Byte_test>
      </Pattern>
      <Pattern pat_id="3">…</Pattern>
      <Pattern pat_id="4">…</Pattern>
      <Pat_Logic>1 AND 2 AND 3 AND 4</Pat_Logic>
   </Patterns>
   <Message>NETBIOS DCEPRC ORPCThis request flood attempt</Message>
   <Session>
      <Session_End>
       <Session_Timeout>60s</Session_Timeout>
       <Session_Pckt_Count>20</Session_Pckt_Count>
       <Session_Threshold>
          <Session_Threshold_Type>both</Session_Threshold_Type>
          <Session_Threshold_Track>dst</Session_Threshold_Track>
       </Session_Threshold>
      </Session_End>
      <Session_Instructions>
        <Session_Case>
          <Direction>sd</Direction>
          <Case_State_Condition>
            <Isset_Var var="dce.isystemactivator.bind.call.attempt" />
          </Case_State_Condition>
        </Session_Case>
      </Session_Instructions>
   </Session>
</Signature>
```

Diagram8 CIDSS stateful signature

Snort rule shown below is equal with CIDSS rule from (Diagram8):

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 135
(msg:"NETBIOS DCEPRC ORPCThis request flood attempt";
flow:to_server,established; content:"|05|"; within:1;
content:"|00|"; within:1; distance:1; byte_test:1,&,1,0,relative;
content:"|05|"; within:1; distance:21; content:"MEOW";
flowbits:isset,dce.isystemactivator.bind.call.attempt;
threshold:type both, track by_dst, count 20, seconds 60;
reference:cve,CAN-2003-0813;
reference:url,www.microsoft.com/technet/security/bulletin/MS04-
011.mspx; classtype:misc-attack; sid:2494; rev:3;)
```

Diagram9 Snort stateful signature

Summary

Signatures standard should be used for improving security of information systems. CIDSS may be used by IT administrators for translating, moving, analyzing different signatures from various manufacturers. There are many other possibilities. CIDSS makes easier to develop analysis and audit tools. Moreover it is possible to develop knowledge bases containing precise and easy-to-implement vulnerabilities information expressed as signatures in common format. Such databases could be maintained by independent developers. They could sell access to most recent and tested signatures selected to fit clients need. Other possibility of use of CIDSS is to make it standard language for self-learning (intelligent) systems used in different categories of information security. It is so special usages, because CIDSS is currently language that has more functionality from Snort.

[1] Internet Engineering Task Force, Internet Draft, Common Intrusion Detection Signatures Standard, http://www.ietf.org

[2] IDS Signature Translator (2005), http://sigtranslator.sourceforge.net

[3] Snort User Manual, http://www.snort.org/docs/