Noah M. Jorgenson

Signature-based IDS with IP packet reassembly support

Project seemed fairly straight forward, there would not be much additional parsing of packets and the majority of what I needed to add was logic that implemented the IDS functionality. Ultimately this came out to be five classes working together on top of the existing packet, parsing and reassembly classes I'd written in projects one and two. The five classes are as follows: RuleParser, IDSRule, IDSRuleOption, CIDR and PortRange. The names are fairly self-explanatory but I will detail my approach and their general function(s). For most there is a main method in which I tested the classes so that I could ensure quickly they were working in the way I expected them to work. For example, the RuleParser is able to take in a filename and spit back out the parsed rules which are constructed as IDSRules. It parses out the rules via some smart string splitting (space delimiters for rule headers, parentheses for options and semicolons for each option). The IDSRule consists of the action to take, protocol, source/left address, source/left port, directionality, destination/right address, destination/right port and finally options. Each IDSRuleOption represented as an identifier and a value, the identifier being the string that is used in the rule/signature (e.g. msg, itype, etc.). There are some auxiliary methods for determining whether or not an option is set, negated, etc. With this setup I found that actually implementing my ruleCheck methods in each packet type, which take the IDSTuple from project 2 and the rule itself, was very straight forward. In addition, the ruleCheck called smaller methods such as 'check_itype' which would simply compare a rule value to the packet's value. I found that most of my checks were one line long and that made me confident I had implemented the logic fairly well. The flags and fragbits option fields were the only methods which were a bit longer, including some clever logic I came up with using ternary if statements and some parsing of the option value. The CIDR and PortRange classes were classes I wrote to simplify the address and port handling and subsequent comparing for each rule. CIDR accepts the string representing either the IP Address/CIDR notation and converts the address into an integer after bit shifting. This integer value is used to compared other IP address and determine if it is within a range given a specified mask. With no mask specified (i.e. a single address), the mask defaults to 32 which will mean just that one IP address is in the CIDR block. The PortRange class I developed takes the port range given in the rule/signature and creates the first and last ports to make up the range. Both CIDR and PortRange have a 'contains' method which will compare either an InetAddress or integer value, respectively, to determine if a value is within the range of that specified in the rule/signature.

I tested my program on the 4 .dat files provided on the course website and was able to construct rules to identify all the attacks. Research was necessary to determine what information in the packets was what I would be constructing the rule around and what I would not. I discuss my rationale briefly below in the sample output for each attack. To run the program simple run the SignatureIDS class via command line.

```
Usage: java SignatureIDS [options...]
Options:
      -r <file>   Filename containing rule definitions/signatures
      -i <file>   Filename containing packet bytes to read in
      -o <file>   Filename to output alerts matching signature
      -c <count>  Exit after receiving count packets
      -a          Append to IDS output log instead of overwriting
```

Winnuke Attack

My IDS found 2 offending packets (of the 8 given), i.e. they contained the URG flag which is used to push data to vulnerable windows machine, sometimes I read it's on port 139 but it does not have to be so I did not include that portion in my rule. You'll notice they repeat, only because to be safe I filter immediately once I see a packet and then again once it's reassembled (even if it's the only fragment in it's set (i.e. first and last fragment).

```
[Wed Apr 20 04:13:26 CDT 2011] winnuke attack (alert ip any any -> any any
(msg: "winnuke attack"; flags: "U+";))
[tcp](60 b) SID: 5 [1 fragments]
Source MAC: 00-01-03-CD-5E-2D
Destination MAC: 00-01-03-CD-50-3C
EtherType: IPv4
Header Length: 20
TOS: 0
Total Length: 43
Data Length: 23
Identification: 1325
FlagFrag Binary: 0100000000000000
Flags: 010
Last Fragment?: true
Fragment Offset: 0
First octet: 0
Last octet: 23
TTL: 64
Payload Start: 34
Protocol: TCP
Source Address: /192.168.1.200
Destination Address: /192.168.1.50
Number of options: 0
Source Port: 32771
Destination Port: 139
Sequence Number: 57569210
Acknowledgement Number: 974037
Control Flags: 00111001
URG Flag: true

[Wed Apr 20 04:13:26 CDT 2011] winnuke attack (alert ip any any -> any any
(msg: "winnuke attack"; flags: "U+";))
[tcp](57 b) SID: 1 [1 fragments]
Source MAC: 00-01-03-CD-5E-2D
Destination MAC: 00-01-03-CD-50-3C
EtherType: IPv4
Header Length: 20
TOS: 0
Total Length: 43
Data Length: 23
Identification: 1325
FlagFrag Binary: 0100000000000000
Flags: 010
Last Fragment?: true
Fragment Offset: 0
First octet: 0
Last octet: 23
TTL: 64
Payload Start: 34
Protocol: TCP
Source Address: /192.168.1.200
Destination Address: /192.168.1.50
Number of options: 0
Source Port: 32771
```

```
Destination Port: 139
Sequence Number: 57569210
Acknowledgement Number: 974037
Control Flags: 00111001
URG Flag: true

[Wed Apr 20 04:13:26 CDT 2011] winnuke attack (alert ip any any -> any any
(msg: "winnuke attack"; flags: "U+";))
[tcp](60 b) SID: 5 [1 fragments]
Source MAC: 00-01-03-CD-5E-2D
Destination MAC: 00-01-03-CD-50-3C
EtherType: IPv4
Header Length: 20
TOS: 0
Total Length: 43
Data Length: 23
Identification: 1323
FlagFrag Binary: 0100000000000000
Flags: 010
Last Fragment?: true
Fragment Offset: 0
First octet: 0
Last octet: 23
TTL: 64
Payload Start: 34
Protocol: TCP
Source Address: /192.168.1.200
Destination Address: /192.168.1.50
Number of options: 0
Source Port: 32771
Destination Port: 139
Sequence Number: 57569210
Acknowledgement Number: 974037
Control Flags: 00111000
URG Flag: true

[Wed Apr 20 04:13:26 CDT 2011] winnuke attack (alert ip any any -> any any
(msg: "winnuke attack"; flags: "U+";))
[tcp](57 b) SID: 1 [1 fragments]
Source MAC: 00-01-03-CD-5E-2D
Destination MAC: 00-01-03-CD-50-3C
EtherType: IPv4
Header Length: 20
TOS: 0
Total Length: 43
Data Length: 23
Identification: 1323
FlagFrag Binary: 0100000000000000
Flags: 010
Last Fragment?: true
Fragment Offset: 0
First octet: 0
Last octet: 23
TTL: 64
Payload Start: 34
Protocol: TCP
Source Address: /192.168.1.200
```

```
Destination Address: /192.168.1.50
Number of options: 0
Source Port: 32771
Destination Port: 139
Sequence Number: 57569210
Acknowledgement Number: 974037
Control Flags: 00111000
URG Flag: true
```

JoltUDP Attack

Looked the same as Jolt2 with a different source address and protocol obviously to me, similar rule checking fragment offset caught it.

```
[Wed Apr 20 02:45:54 CDT 2011] jolt udp attack (alert udp any any -> any any
(msg: "jolt udp attack";))
[udp](60 b) SID: 4 [1 fragments]
Source MAC: 00-01-03-CD-5E-2D
Destination MAC: 00-01-03-CD-5E-15
EtherType: IPv4
Header Length: 20
TOS: 0
Total Length: 29
Data Length: 9
Identification: 1109
FlagFrag Binary: 0001111111111110
Flags: 000
Last Fragment?: true
Fragment Offset: 8190
First octet: 65520
Last octet: 65529
TTL: 255
Payload Start: 34
Protocol: UDP
Source Address: /192.168.1.20
Destination Address: /192.168.1.10
Number of options: 0
Source Port: 1235
Destination Port: 80
```

Jolt2 Attack

After looking up the attack I found that the packet was illegal, wrapping the size limit, and had a reported offset of 65520, which would make the field 65520/8 or, 8190. A simple rule check for the fragment offset value caught the packet in my IDS.

```
[Wed Apr 20 02:42:44 CDT 2011] jolt2 attack (alert icmp any any -> any any
(msg: "jolt2 attack";))
[icmp](60 b) SID: 4 [1 fragments]
Source MAC: 00-01-03-CD-5E-2D
Destination MAC: 00-01-03-CD-5E-15
EtherType: IPv4
Header Length: 20
TOS: 0
Total Length: 29
Data Length: 9
Identification: 1109
```

Noah M. Jorgenson

```
FlagFrag Binary: 0001111111111110
Flags: 000
Last Fragment?: true
Fragment Offset: 8190
First octet: 65520
Last octet: 65529
TTL: 255
Payload Start: 34
Protocol: ICMP
Source Address: /192.168.1.200
Destination Address: /192.168.1.10
Number of options: 0
Type: 8
Code: 0
```