Identifying Brute Force Host Access Attempts

Joe McManus, joe@inotion.com
Aaron Shelmire, shelmire@psc.edu
December 5, 2008

Introduction

In our research we have studied the flow of traffic in an attempt to identify typical attack patterns, the successful attacks, and the resulting behavior of the hosts that have been attacked.

Many studies have performed work on vertical-scans and horizontal-scans. This sort of scanning activity focuses on port-scanning although using different methods. A vertical scan will scan several ports on a single host and a horizontal scan would be scanning one port on many hosts. Our study is focusing on the actions that occur when the service has already been found to exist. At this point the attacker will either try to exploit a known vulnerability in the protocol, or will attempt to gain access through poorly-protected account credentials.

We have focused on the following port/protocol pairs:

- 22/ssh
- 23/telnet
- 143,220,585,993/imap

These ports have been studied as the services that run on them require username/password authentication and due to their popularity.

Attack Methods

Our study focuses on two similar attack methodologies. We are interested in finding "brute force" password attacks. Password guessing attacks consist of two kinds random character and dictionary attacks. Dictionary attacks are a simple attack which attempt to gain access to a system through combinations of well known usernames passwords (root/toor, guest/guest). Brute force attacks will use random characters for passwords, typically with a list of known user accounts in an attempt to exhaust all the possible passwords the user might have.

Dictionary Attacks

Dictionary Attacks consist of connecting to a service and attempting common usernames and passwords. Attackers attempt to guess usernames based on common service accounts e.g. apache, mysql, oracle, root, etc. These accounts are often configured as part of a base unix or linux install or installed by administrators in an attempt to trouble shoot a service. Additionally attackers will attempt common account usernames such as john, jsmith, johnsmith as is common with how spam addresses are generated.

Often times after an account is set up an administrator may set the username and password to be the same, or based on a dictionary word. These accounts and passwords are well known to attackers, and are often employed in Password Guessing attacks.

These attacks follow a known pattern, and are easy to spot through system logs. The attackers have lists of account names with corresponding passwords. A program then iterates through the list trying each one of these account/password combinations. This behavior will result in many connections. The vast majority of these attempts will fail. Password Guessing attempts may be identified by looking for many repetitions of the normal traffic patterns for failed authentication attempts between a pair of hosts.

Brute Force Attacks

Brute Force attacks are very similar to Dictionary attacks, although they differ in a few important ways.

Brute Force attacks also focus on services that require presenting an account identifier and an associated credential. These attacks often use the same username and password based credentials that are used in Password Guessing attacks, although these attacks are not limited to passwords and are often used against cryptographically generated credentials.

Brute Force attacks are typically performed against known account names. Instead of using only commonly used passwords, brute force attacks will try many combinations, in an attempt to guess all possible combinations.

From the perspective of network flow data the attacks look the same. As with Password Guessing attacks, Brute Force attacks have many failures and can be identified by looking for repeated traffic of failure attempts.

Typical brute force attempts begin with a scan of a hosts looking for open ports. After these ports are discovered a scan may come from the scanning host. However firewall techniques like those listed above will often block these attempts. The attackers will sometimes scan with one host and launch the brute force attack with another. This complicates attacker discovery in flow data as you cannot assume every scanner will launch a brute force attack.

A Case for Flow Data

As stated before we are studying the attempts of attackers to gain access to services through brute-force and password-guessing attacks. These attacks are easy to identify through system log files, but it is often hard to centralize the collection of all log files in an organization. Many system administrators may build new computers and simply not configure the log file mechanisms to send these messages to the central location. Other times a user, perhaps a visiting employee from a partner company, may bring in a laptop to use on the organizations network.

Network flow data is much easier for an organizations staff to control. There are a finite amount of well-known entry/exit points for the networks under an organization's assets. If a way to identify these attacks could utilize this flow data were to be created, it could ease the administrative burden of the organization.

However, there are well known patterns that allow identification of brute-force and password guessing attacks through log analysis. With log analysis a password guessing attack is typified by a few attempts at one username, followed by a few attempts at another username. The password guessing attack will have many usernames that are not valid. A brute-force attack will focus on a few usernames, and will have many attempts against those usernames.

This differs from the view of the attacks seen with flow data. Still the brute-force and password guessing attacks have certain characteristics that allow identification from the flow data alone. These characteristics will affect the way the flow looks. In both cases the two types of attacks will have the same appearance through the flow data. Both forms of attack are typified by many failed authentication attempts. In some instances a successful authentication attempt will occur. The exact details of how this view of the flow will change are determined by the protocol, but the general concept is the same.

There are many open source plug-ins for host based firewalls such as IP Tables that will lock out a system after multiple session attempts on connections to port 22/ssh from the same IP address over a short period of time. Flow analysis helps to pick up these attempts as we can look over long periods of time to determine if a host is attempting to compromise a system.

Protocol Behavior

While all three protocols require authentication through the presentation of account identifiers and secret credentials, each has slightly different behavior. The differentiation of these behaviors is described through the protocol specification.

SSH brute-force/password-guessing attempts

The Secure Shell (SSH) protocol is the most common method of allowing remote shell based access to systems. SSH is a protocol that utilizes encryption upon the network traffic. The encrypted data makes the traffic hard to analyze with traditional Network Intrusion Detection Systems, as the content portion of the packets is rendered useless.

SSH servers have some characteristics that ease network flow analysis. Most SSH servers only allow a finite amount of password attempts before dropping the network connection requiring an attacker to re-establish the connection, meaning a new flow will be established. The default configuration for SSH servers allows only 5 attempts before dropping the connection.

A valid SSH connection with authentication will require more than 14 packets. The first 3 packets are the typical 3-way handshake of SYN, SYN-ACK, ACK. Following this each side will send an identification banner resulting in packets 4 and 5. At a minimum each side will then send a packet in an attempt to negotiate the encryption algorithm to be used bringing the total number of packets to 7. After the encryption algorithm has been negotiated an encryption key must be generated by the SSH_MSG_KEXINIT packets from each side resulting in a minimum total of 9 packets. The server and client must agree upon the keys, and will then generate the actual symmetric keys to use for the session exchanging them with the SSH_MSG_NEWKEYS packets, adding another 2 packets for a minimum total of 11. It's at this point that the client requests an SSH service via the SSH_MSG_SERVICE_REQUEST packet. If the client receives a SSH_MSG_SERVICE_ACCEPT packet from the server the connection will continue. This brings the total number of packets to 13, and allows the SSH Authentication protocol to start. [2]

The SSH Authentication Protocol will begin with the server sending an SSH_USERAUTH_REQUEST. If we assume the first of these packets will be using the service_name of "password" the client can finally respond with a password string. Finally the SSH server would respond with either a SSH_MSG_USERAUTH_SUCCESS or SSH_MSG_USERAUTH_FAILURE packet. Before sending that final packet, the minimum number of packets having traversed between the client and server numbers 15. [1]

A network flow is typically thought of as only one side of the total network connection, resulting in a minimum of 7 packets as part of a valid flow in which a password has been sent. In actuality there will be more than 7 packets per side for a single authentication attempt. These other packets include packets such as SSH2_MSG_KEX_DH_GEX_REQUEST(1024<1024<8192), SSH2_MSG_KEX_DH_GEX_INIT and SSH2_MSG_KEX_DH_GEX_REPLY.

Each password attempt will be sent as a separate packet, with a corresponding failure or success notification received from the server. Due to the characteristic of many failures, the SSH server will reset the connection more often than establishing a successful authenticated connection. Since there will be a small amount of password attempts the flows will not require many packets. Assuming that most SSH servers allow a less than 10 password attempts this type of traffic will take less than 30 packets per flow. These characteristics will result in many flows originating from client hosts attempting to brute force the servers. Additionally, because the amount of attempted connections will remain static, the many flows associated with a client-server pair will be approximately the same length in bytes.

Most SSH traffic is highly user driven. This means that the traffic will behave in a very fluid manner, varying in packet size over time. Traffic that has a constant rate is more indicative of a mechanical process, which would not be user driven. User driven traffic will also have some gaps where only a few packets may be sent.

One conflict that presents itself is the case of SCP connections with small files. This traffic will actually be very different from the authentication attempts. SCP traffic will result in packets approaching the Maximum Transmission Unit (MTU). The MTU of this traffic will either be 1500 bytes or 9000 bytes, both of which are much larger than any password attempts that would be found.

This allows us to create a prototype for SSH brute-force and password guessing attacks. Client-Server pairs engaging in this behavior will have many flows of a few packets (less than 30), with a small amount of bytes being transmitted.

IMAP brute-force/password-guessing attempts

The IMAP protocol is a very popular protocol in use to read corporate email. It began use as an unencrypted protocol, but has more recently been used in an TLS encrypted form. This also makes the traffic generated hard to profile by deep-packet inspection Intrusion Detection Systems.

Popular IMAP clients usually only allow a few password attempts before closing the network connection. An attacker would typically use their own purpose built client for these attacks. These clients would not reset the connection. It may be possible to tune an IMAP server to timeout after a predetermined amount of authentication attempts, but the default seems to not utilize this feature. Our testing showed that a TLS encrypted IMAP flow from a client to a server requires a minimum of 9 packets to instantiate the session, and a following 2 packets to complete a LOGIN attempt that has failed [See Appendix 1].

From testing done by us, we have noticed that typical IMAP clients have a very typical behavior. A typical IMAP client will begin with a flow of many packets. This will correspond with the initial authentication attempts, and then the retrieval of the mailbox from the server. After this initial heavy flow, there will be a lull in traffic. After a time period, the client will again check the mailbox, resulting in another flurry of packets 10 or more packets. This will vary depending upon how much mail has been sent to the inbox, but the time period will be static. Typical time periods seem to be 1 minute. This will result in burst-type traffic.

A brute-force or password guessing attempt will exhibit different traffic patterns. The flow of traffic will remain relatively constant, with either a few packets sent every few seconds or a constant stream of many packets being sent. This is indicative of a mechanized process. However, all IMAP servers are mechanized processes. The difference is in how most mail clients are configured. They will typically have flows that have bursty traffic. These bursts will occur every time the client checks for more messages. Typically a pause for a few minutes will occur, followed by a lot of traffic as opposed to the brute-force or password guessing programs constant stream of packets. Additionally the size of the packets in the traffic of brute-force and password guessing

programs will remain approximately constant, and will be smaller in size since only a few bytes will be required for the IMAP protocol.

Telnet brute-force/password-guessing attempts

Our third protocol, Telnet, provides an unencrypted login directly to the command line shells. At one time Telnet occupied the space that SSH currently does. However, due to the unencrypted data flow, many organizations abandoned the use of Telnet.

Traditional Telnet connections are the perfect candidate for deep-packet inspection Intrusion Detection Systems. More recent versions of Telnet have been made using encryption schemes such as Kerberos, removing the usefulness of deep-packet inspection.

The Telnet protocol provides a unique challenge for analysis via network flow data. A Telnet connection stays open while the authentication attempts are made. The server may be configured to only allow a certain amount of authentication attempts, but the server will typically cycle within the network connection, not requiring the client to re-establish the connection.

Contrasting a typical Telnet session with a brute-force or password-guessing session may be the only manner in which to identify such attacks. This still proves to be quite challenging. A typical Telnet session will be user driven. This would result in an initial burst of activity, followed by a constant but varying amount of activity. Brute-force and password-guessing attacks will result in a stream of activity at a nearly constant rate.

Our Findings

We have already performed a thorough analysis of anonymized traffic for 20 days in the fall of 2008 involving the IMAP and SSH protocols. Analysis of the Telnet protocol was only touched on. Our models have had to vary considerably since the traffic is sampled at the very low rate of 1 out of every 100 packets. In addition to the sampling rate, the length of each flow is 60 seconds.

IMAP

With the IMAP traffic we have initially filtered based upon source and destination IP-address pairs. We immediately drop all pairs that have less than 5 flows, as it is our experience that a brute force or password guessing attack will last significantly longer than 5 minutes. Each of those source and destination IPs are added to a list, and then used as criteria for more filtering.

Due to the sampling characteristics of our data, the flows may have gaps were traffic is not seen by the sensor for a few minutes, yet if we weren't sampling the packets would appear and generate a flow for that time period. It is because of this characteristic that we drop all host/destination IP pairs that have gaps of more than 10 minutes. This is to allow

the sensor to miss the packets for 10-minutes of time. At this point we may still have traffic from hosts that check mail in a time-span of less than 10 minutes.

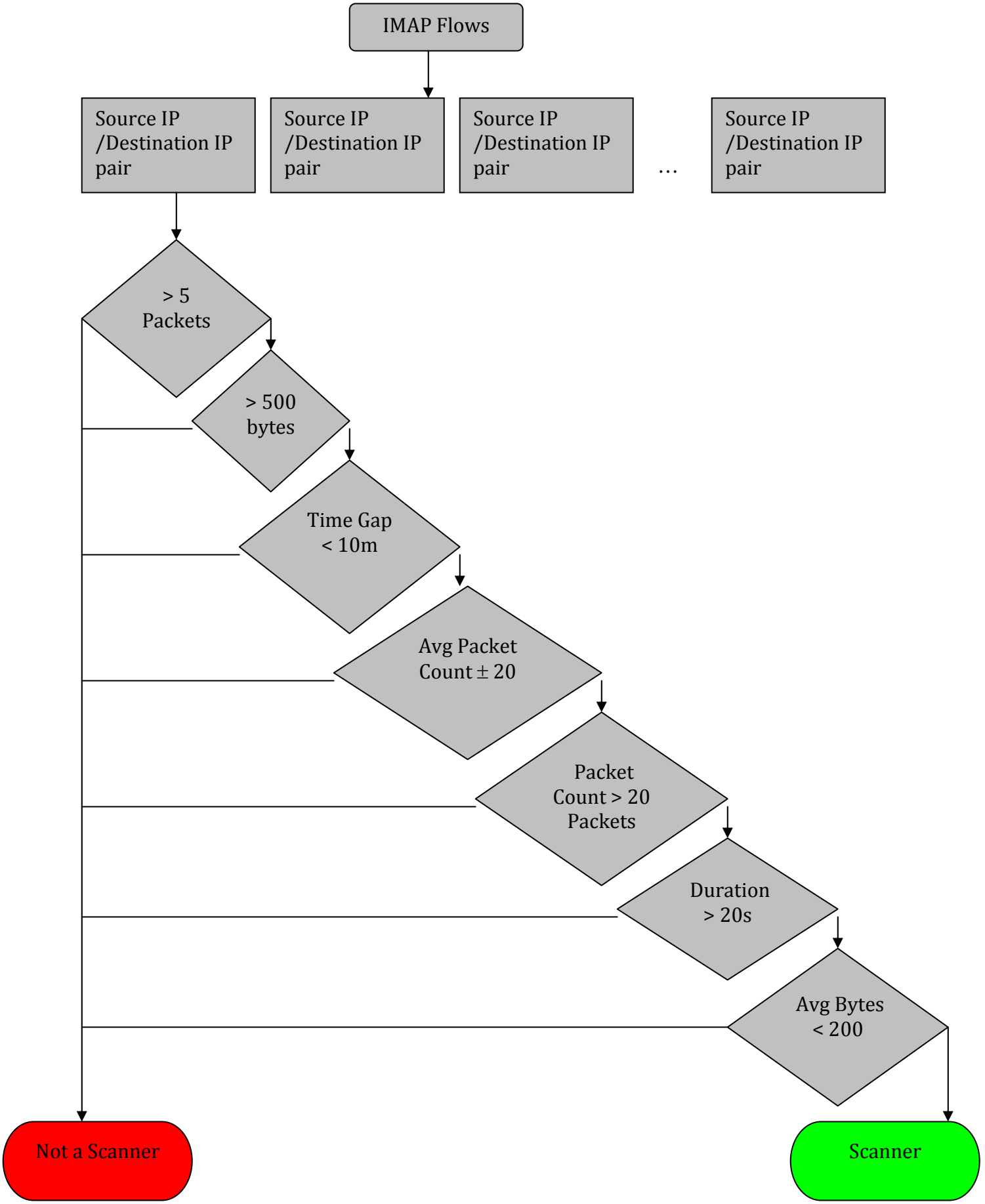
The next criteria we use to drop packets is based upon the difference in the amount of packets between flows. If we have more than 3 flows that vary by more than 20 packets we drop the source/destination pair from consideration.

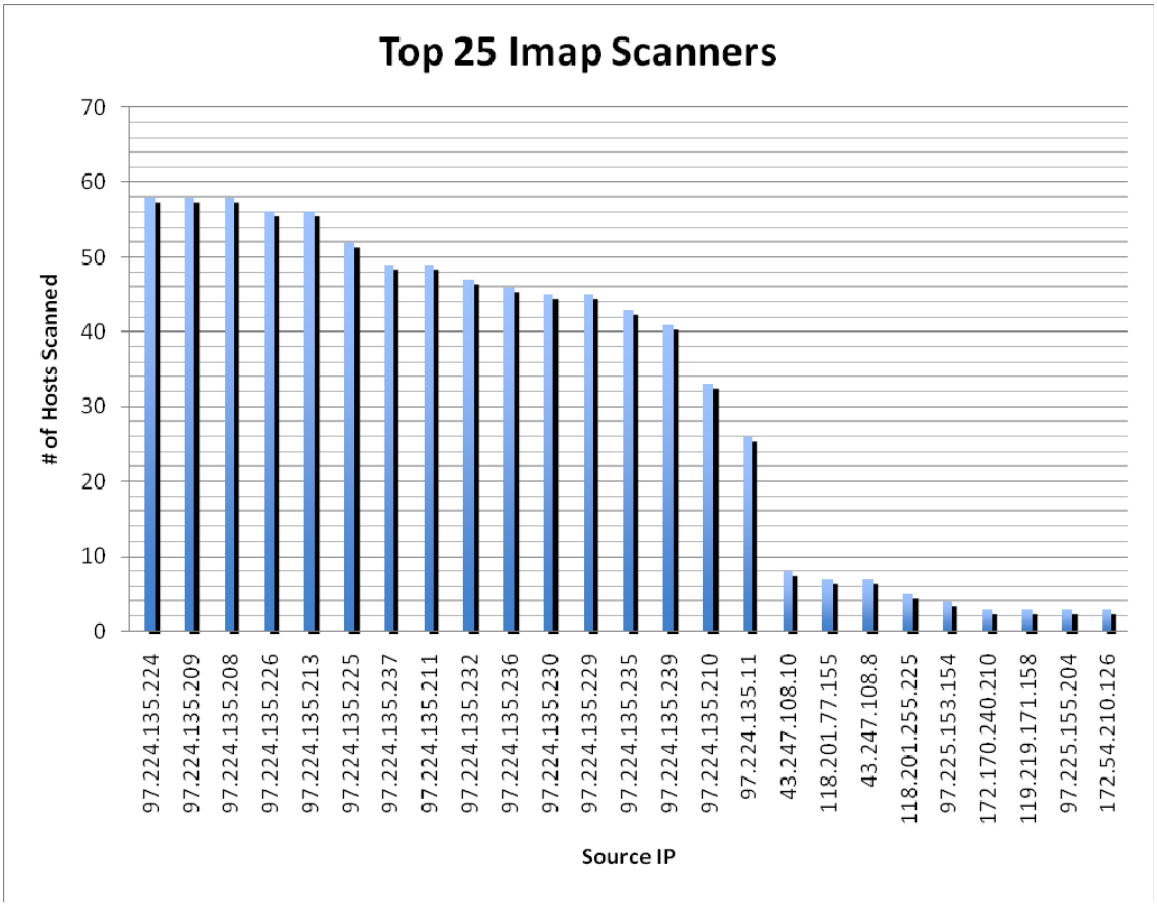
We next sum the total number of packets seen by the filter. A brute-force or password guessing attack will generate a fair amount of traffic, more than likely more than 6000 packets. If this sum is less than 60 packets at the 1 out of 100 sampling rate, we drop the source/destination pair from consideration.

Then we look at the duration times in the flow records. The IMAP protocol returns from a failed authorization attempt very quickly. This allows the brute-force/password guessing process to send a lot of traffic, resulting in the sensor catching many packets and enabling it to generate duration timings. Most of the flows from these automated processes will last for the entire duration of our flow time(60 seconds). Due to this behavior we drop source/destination pairs that have many flows lasting less than 20 seconds.

Finally we check the average byte count of the flows. Sending simple LOGIN command with a username and password does not require many bytes. If the average byte count is above 200 bytes, we drop the source and destination pair from consideration.

In the flow data that we have this leaves us with a small list of source/destination IP pairs to check for each day.


IMAP Flows
Source IP /Destination IP pair
Source IP /Destination IP pair
Source IP /Destination IP pair
…
Source IP /Destination IP pair
> 5 Packets
> 500 bytes
Time Gap < 10m
Avg Packet Count ± 20
Packet Count > 20 Packets
Duration > 20s
Avg Bytes < 200
Not a Scanner
Scanner

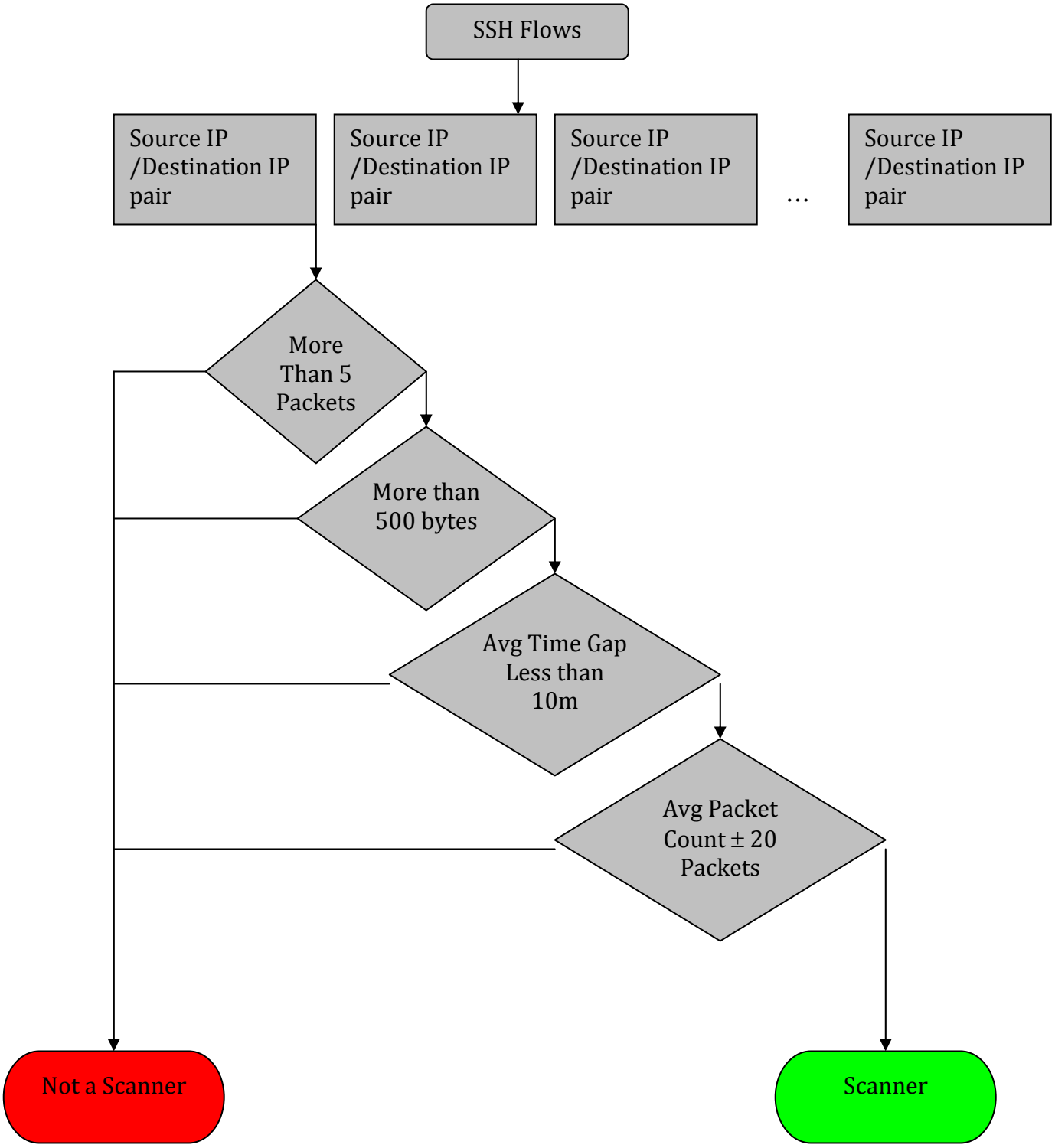


Of these scanning hosts none of them had any data to let us believe they had been compromised. However this data shows that the anonymized traffic that is 97.224.135 should be blocked at your perimiter.

SSH

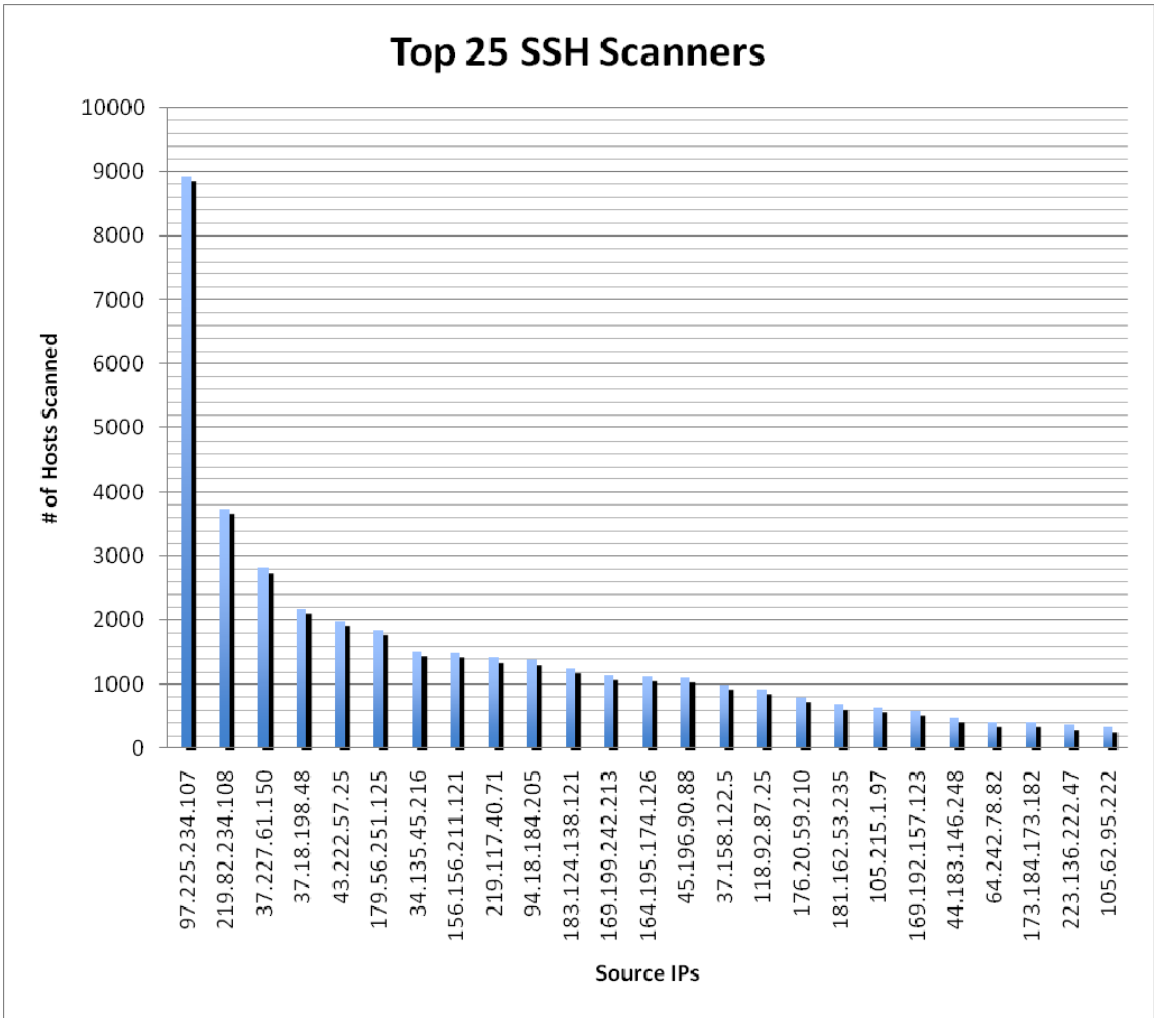
Much of the analysis performed on the IMAP protocol translated to the SSH protocol. In both instances you are looking for a mechanized process that is not typical of the traffic associated with normal use of the protocol. We filtered each flow based upon source and destination IP address. We also immediately dropped all source and destination IPs that had less than 5 flow records.

The majority of traffic that appears in brute force/password guessing attacks upon SSH, most traffic will be authentication attempts. The authentication attempt packets will be much smaller than typical traffic. Our first instance of filtering utilizes this characteristic to remove source and destination IP pairs that have average byte counts above 500.


SSH Flows
Source IP /Destination IP pair
Source IP /Destination IP pair
Source IP /Destination IP pair
…
Source IP /Destination IP pair
More Than 5 Packets
More than 500 bytes
Avg Time Gap Less than 10m
Avg Packet Count ± 20 Packets
Not a Scanner
Scanner

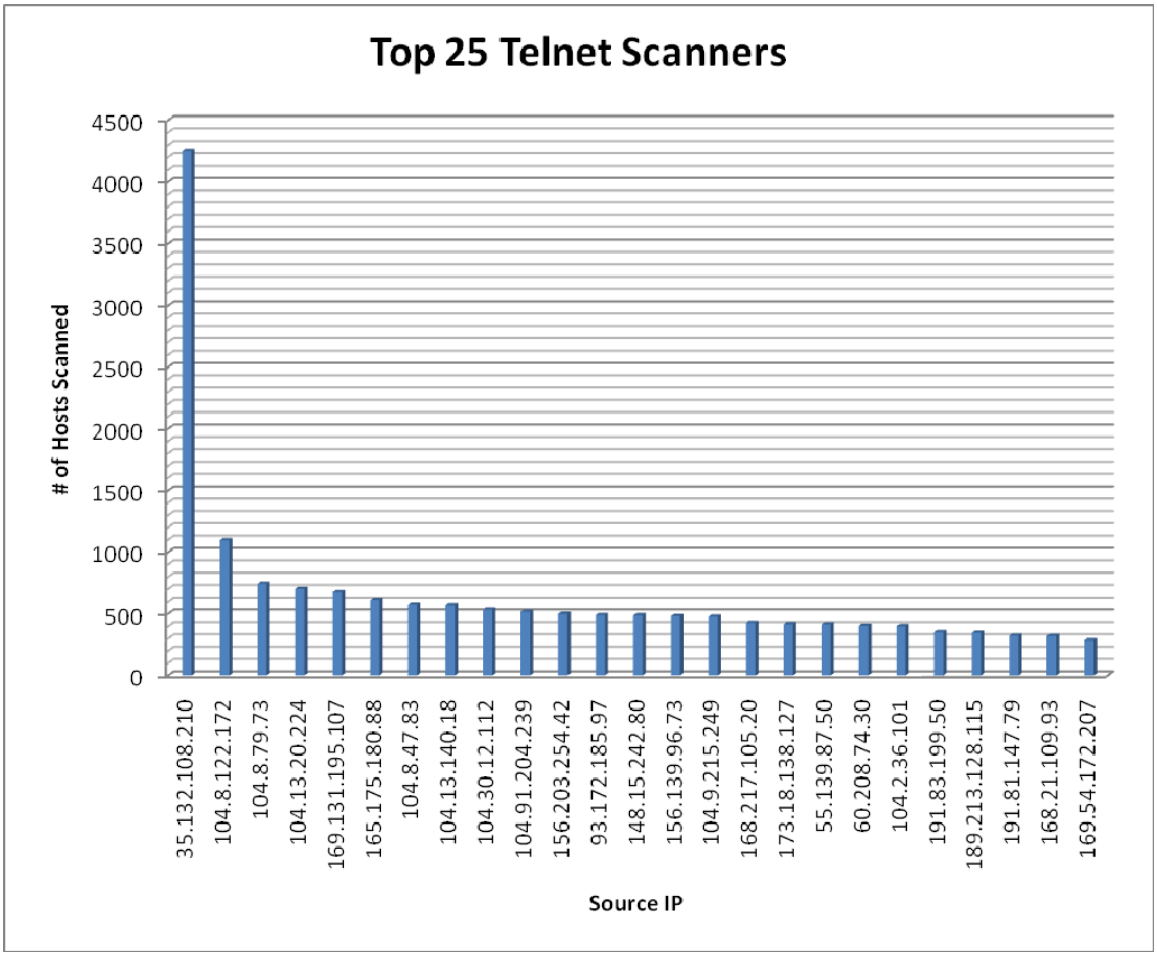
A mechanized brute-force/password guessing attack will be mechanized with a nearly constant rate of data transmission. This will result in flows with very little gaps. We next remove source/destination IP pairs that have gaps of 10 minutes or more between flows over 24 hours. We also remove flows that have a difference of more than 20 packets between flows.

The SSH protocol has a much slower authentication process than IMAP, resulting in less packets sent over 60 seconds. This aspect along with the 1 out of 100 sampling rate makes analysis incredibly difficult. The sensor usually only catches 1 packet per flow, and sometimes misses a source/destination pairs flow for that minute, even when there has been traffic. Having a smaller sampling rate and a longer flow length would enable much more accurate analysis of the data.



Telnet

We believe the same methods of discovery as listed above can be used to determine telnet attacks. We need to continue our work in this area but do not foresee any problems. We have done some initial work which shows that telnet scanning is more popular than imap scanning. However no further work was completed.



Conclusion

The IMAP and telnet analysis showed no hosts compromised. The SSH analysis showed that 2 hosts communicated back to the scanning hosts more than the minimum number of packets. However with sampled data we have a problem of not seeing the entire conversation between attacker and target.

What we would like to do going further would be to analyze the traffic of the compromised hosts after the compromise has happened. It would be interesting to see if

they have become scanning hosts, used as malware hosts or bots for spam. To then study the traffic from attacked hosts and automate the process so that when hosts are detected that have been compromised alerts are sent off. Traffic is automatically studied and plugins for firewall scripts like the ones mentioned above could be developed to drop the machine off the network.

Appendix 1

The following are the results of watching the client side communications through tcpdump while connecting to a TLS enabled IMAP server.

Tcpdump

tcpdump -vvv -i en1 dst host 192.168.66.176

tcpdump: listening on en1, link-type EN10MB (Ethernet), capture size 96 bytes

11:44:19.163400 IP (tos 0x0, ttl 64, id 29977, offset 0, flags [DF], proto TCP (6), length 64) 10.100.101.138.56079 > 192.168.66.176.imaps: S, cksum 0xe262 (correct), 3173928557:3173928557(0) win 65535 <mss 1460,nop,wscale 3,nop,nop,timestamp 148516321 0,sackOK,eol>

11:44:19.182675 IP (tos 0x0, ttl 64, id 14554, offset 0, flags [DF], proto TCP (6), length 52) 10.100.101.138.56079 > 192.168.66.176.imaps: ., cksum 0x514c (correct), 3173928558:3173928558(0) ack 1371966505 win 65535 <nop,nop,timestamp 148516321 2689487509>

11:44:19.247195 IP (tos 0x0, ttl 64, id 8500, offset 0, flags [DF], proto TCP (6), length 170) 10.100.101.138.56079 > 192.68.66.176.imaps: P 0:118(118) ack 1 win 65535 <nop,nop,timestamp 148516322 2689487509>

11:44:19.268360 IP (tos 0x0, ttl 64, id 12148, offset 0, flags [DF], proto TCP (6), length 52) 10.100.101.138.56079 > 192.168.66.176.imaps: ., cksum 0x4748 (correct), 118:118(0) ack 2509 win 65389 <nop,nop,timestamp 148516322 2689487592>

11:44:19.333460 IP (tos 0x0, ttl 64, id 19257, offset 0, flags [DF], proto TCP (6), length 378) 10.100.101.138.56079 > 192.168.66.176.imaps: P 118:444(326) ack 2509 win 65535 <nop,nop,timestamp 148516323 2689487592>

11:44:19.377461 IP (tos 0x0, ttl 64, id 5422, offset 0, flags [DF], proto TCP (6), length 52) 10.100.101.138.56079 >192.168.66.176.imaps: ., cksum 0x44c4 (correct), 444:444(0) ack 2568 win 65535 <nop,nop,timestamp 148516323 2689487704>

11:44:19.395663 IP (tos 0x0, ttl 64, id 4103, offset 0, flags [DF], proto TCP (6), length 52) 10.100.101.138.56079 > 192.168.66.176.imaps: ., cksum 0x43dc (correct), 444:444(0) ack 2781 win 65535 <nop,nop,timestamp 148516323 2689487723>

At this point the initial TLS handshake has occurred. The next two packets are sent when the "AAA Login username passwd" command is sent.

11:44:50.340356 IP (tos 0x0, ttl 64, id 25236, offset 0, flags [DF], proto TCP (6), length 142) 10.100.101.138.56079 > 192.168.66.176.imaps: P 444:534(90) ack 2781 win 65535 <nop,nop,timestamp 148516633 2689487723>

11:44:53.365874 IP (tos 0x0, ttl 64, id 27647, offset 0, flags [DF], proto TCP (6), length 52) 10.100.101.138.56079 > 192.168.66.176.imaps: ., cksum 0xbd30 (correct), 534:534(0) ack 2850 win 65535 <nop,nop,timestamp 148516663 2689521699>

The next 4 packets are the next two attempts.

11:45:19.628748 IP (tos 0x0, ttl 64, id 15313, offset 0, flags [DF], proto TCP (6), length 142) 10.100.101.138.56079 > pscuxb.psc.edu.imaps: P 534:624(90) ack 2850 win 65535 <nop,nop,timestamp 148516925 2689521699>

11:45:22.653593 IP (tos 0x0, ttl 64, id 32344, offset 0, flags [DF], proto TCP (6), length 52) 10.100.101.138.56079 > 192.168.66.176.imaps: ., cksum 0x48fd (correct), 624:624(0) ack 2919 win 65535 <nop,nop,timestamp 148516956 2689550994>

11:45:36.318361 IP (tos 0x0, ttl 64, id 4976, offset 0, flags [DF], proto TCP (6), length 142) 10.100.101.138.56079 > 192.168.66.176.imaps: P 624:714(90) ack 2919 win 65535 <nop,nop,timestamp 148517092 2689550994>

11:45:39.345954 IP (tos 0x0, ttl 64, id 43333, offset 0, flags [DF], proto TCP (6), length 52) 10.100.101.138.56079 > 192.168.66.176.imaps: ., cksum 0x0683 (correct), 714:714(0) ack 2988 win 65535 <nop,nop,timestamp 148517123 2689567686>

The next four packets are actually concerned with closing the connection when the 3 minute timeout is received.

11:48:39.342198 IP (tos 0x0, ttl 64, id 47886, offset 0, flags [DF], proto TCP (6), length 52) 10.100.101.138.56079 > 192.168.66.176.imaps: ., cksum 0x3fe7 (correct), 714:714(0) ack 3057 win 65535 <nop,nop,timestamp 148518922 2689747731>

11:48:39.343283 IP (tos 0x0, ttl 64, id 60412, offset 0, flags [DF], proto TCP (6), length 52) 10.100.101.138.56079 > 192.168.66.176.imaps: ., cksum 0x3fe5 (correct), 714:714(0) ack 3058 win 65535 <nop,nop,timestamp 148518922 2689747732>

11:48:39.358642 IP (tos 0x0, ttl 64, id 38801, offset 0, flags [DF], proto TCP (6), length 89) 10.100.101.138.56079 > 192.168.66.176.imaps: P 714:751(37) ack 3058 win 65535 <nop,nop,timestamp 148518922 2689747732>

11:48:39.358775 IP (tos 0x0, ttl 64, id 54495, offset 0, flags [DF], proto TCP (6), length 52) 10.100.101.138.56079 > 192.168.66.176.imaps: F, cksum 0x3fbf (correct), 751:751(0) ack 3058 win 65535 <nop,nop,timestamp 148518922 2689747732>

^C

17 packets captured

349 packets received by filter

0 packets dropped by kernel

Connection Commands

```
openssl s_client -connect 192.168.66.176:993

CONNECTED(00000003)

depth=2
/C=US/ST=Astate/L=City/O=Organization/OU=SomeBranchofOrg/CN
=Org CA Server

verify error:num=19:self signed certificate in certificate
chain

verify return:0

---

Certificate chain

 0
s:/C=US/ST=Astate/L=City/O=Organization/OU=SomeBranchofOrg/
CN =192.168.66.176


i:/C=US/ST=Astate/L=City/O=Organization/OU=SomeBranchofOrg/
CN=Org CA web 2

1
s:/C=US/ST=Astate/L=City/O=Organization/OU=SomeBranchofOrg/
CN=Org CA web 2


i:/C=US/ST=Astate/L=City/O=Organization/OU=SomeBranchofOrg/
CN=OrgRoot CA server 2

2
s:/C=US/ST=Astate/L=City/O=Organization/OU=SomeBranchofOrg/
CN=OrgRoot CA server 2


i:/C=US/ST=Astate/L=City/O=Organization/OU=SomeBranchofOrg/
CN=OrgRoot CA server 2

---
```

```
Server certificate

-----BEGIN CERTIFICATE-----

+hlfa8r52+u+gcYbgEHbqMNXq4oZuh7hifuRA/2ftqHTGzwkh9ohs7

-----END CERTIFICATE-----

subject=/C=US/ST=Astate/L=City/O=Organization/OU=SomeBranch
ofOrg/CN =192.168.66.176

issuer=/C=US/ST=Astate/L=City/O=Organization/OU=SomeBrancho
fOrg/CN=Org CA web 2

---

No client certificate CA names sent

---

SSL handshake has read 2567 bytes and written 444 bytes

---

New, TLSv1/SSLv3, Cipher is AES256-SHA

Server public key is 2048 bit

SSL-Session:

    Protocol  : TLSv1

    Cipher    : AES256-SHA

    Session-ID:

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Session-ID-ctx:

    Master-Key: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

    Key-Arg   : None

    Start Time: 1223912659

    Timeout   : 300 (sec)

    Verify  return  code:  19  (self  signed  certificate  in
certificate chain)

---
```

```
* OK [CAPABILITY IMAP4REV1 I18NLEVEL=1 LITERAL+ SASL-IR
LOGIN-REFERRALS AUTH=GSSAPI AUTH=PLAIN AUTH=LOGIN]
192.168.66.176 IMAP4rev1 2007b.404 at Mon, 13 Oct 2008
11:44:19 -0400 (EDT)

AAA Login username passwd

AAA NO Invalid login credentials

AAA Login username passwd

AAA NO Invalid login credentials

AAA Login username passwd

AAA NO Invalid login credentials

* BYE Autologout (idle for too long)
```

References

1, Ylonen, T. “The Secure Shell (SSH) Authentication Protocol”, Request for Comments: 4252, January 2006.

2. Ylonen, T. “The Secure Shell (SSH) Transport Layer Protocol”, Request for Comments: 4253, January 2006.

3. Crispin, M “Internet Message Access Protocol - Version 4rev1”, Request for Comments: 3501, March 2003

4. Newman, C “Using TLS with IMAP, POP3, and ACAP”, Request for Comments: 2595, June 1999