Analyzing the Effect of HTTP Port Configuration on Attacker Behavior

University of Maryland, College Park

HACS202 - Spring 2020 (Group A)

Data Analysis + Notebook Author:

Akilesh **Praveen**

Groupmates:

Harlon **Dobson**, Ian **Parks**, Julie **Yang**

This notebook showcases the scraping of data from honeypot logs, the tidying of said data, and the interpretation of this data. This repository also contains the raw scripts used to perform data scraping, with additional debug features included. Find them in the 'scripts' directory.

Table of Contents

Section 0: Overview

This project was our attempt at answering the research question, "Does HTTP/HTTPS port configuration influence attacker behavior on a system?". We leveraged honeypots in order to solve this key question.

We let our honeypots run for one week in order to collect attacker data and come to reasonable conclusions. My groupmates handled the setup of the honeypot infrastructure on the UMD ACES infrastructure, and I was tasked with data extraction and interpretation.

Disclaimer: We understand that this data can be labeled as inconclusive due to the short amount of time we collected it over. Regardless, this analysis is based on the data we have collected.

Below, we include the libraries and initialization code for the project.

In [1]:
```
import re

## Data structures
import pandas as pd

## Import data visualization packages
import matplotlib.pyplot as plt
%matplotlib inline
```

Section 1: Data Collection

Supplementary Dataset: Detailed Attack Logs

First, let's extract the data from the `mitm_files` folder. This script is a little involved, but it essentially separates the log into 'attacks'. Each attack is considered begun when the script detects a connection initiated, and each attack is considered ended when the script detecs that the connection is closed. From this, we extract the date & time of the attack, whether a password was used, the IP address of the attacker, and the client that the attacker was using.

This data was then transferred from a list of lists to a dataframe.

When connections were established and then not closed, that data appeared as ambiguous, and was therefore excluded. Due to the complex and erratic nature of the way event storage was handled in this file, meaningful results in this category were only collected for containers 102 and 103.

Container 102: HTTPS

In [2]:
```
def analyze_attack(fp):
    line = fp.readline()

    if not line:
        return None

    # read the first line of an attack
    initial = re.search("([0-9]{4}-[0-9]{2}-[0-9]{2})\s([0-9]{2}:[0-
9]{2}:[0-9]{2})\.[0-9]{3}\s-\s\[Debug\]\s\[Connection\]\sAttacker\sco
nnected:\s(.*)\s\|\sClient\sIdentification:\s(.*)", line)

    if initial:
        my_date = initial.group(1)
        my_time = initial.group(2)
        my_ip = initial.group(3)
        my_client = initial.group(4)

        # number of attempts + whether a password was used or not
        my_attempts = 0
        my_passwords = []

        # this attack continues until we find the 'Attacker closed th
e connection' line
        end = False
        while(not end):
            nextline = fp.readline()
            found_end = re.search("Attacker\sclosed\sthe\sconnection
$",nextline)

            if found_end:
                end = True
                return [my_date, my_time, my_ip, my_client, my_attemp
ts, my_passwords]

            # this line is not the end, we can get data from it

            # check to see if password attempts have increased
            attempt_add = re.search("has\sso\sfar\smade", nextline)

            if attempt_add:
                my_attempts += 1

            password_add = re.search("trying\sto\sauthenticate\swith
\s\"(.*)\"", nextline)

            if password_add:
                if password_add.group(1) == "none":
                    my_passwords.append(False)
                else:
                    my_passwords.append(True)
```

```
    else:
        # not a distinct 'attack'- move to next event
        read_until_next(fp)
        return [None, None, None, None, None, None]


def read_until_next(fp):
    line = fp.readline()

    while (line):
        nextgroup = re.search("Attacker\sclosed\sthe\sconnection$",li
ne)

        if nextgroup:
            break
        else:
            line = fp.readline()

# use analyze_attack to store all attacks in a list for now


mitm_data_102 = []
mitm_data_103 = []

# open all 3 files and analyze them

with open('../raw_data/mitm_files/mitm_file102', 'r') as fp:
    # fp is now our file pointer to the mitm data file.

    # skip the five lines at the top
    line = fp.readline()
    line = fp.readline()
    line = fp.readline()
    line = fp.readline()
    line = fp.readline()


    while line:
        curr = analyze_attack(fp)
        if curr is None:
            break
        else:
            mitm_data_102.append(curr)



# move the contents of data to a dataframe
mitm_df_102 = pd.DataFrame(mitm_data_102, columns = ['date', 'time',
'ip_address', 'client_type', 'attempts', 'password_used' ])
mitm_df_102
```

Out[2]:

	date	time	ip_address	client_type	attempts	password_used
0	2020-05-04	15:14:00	195.3.147.47	SSH-2.0-paramiko_1.8.1	1.0	[False, True]
1	2020-05-04	15:14:08	195.3.147.47	SSH-2.0-OpenSSH_5.3	1.0	[False, True]
2	2020-05-04	15:14:20	195.3.147.47	SSH-2.0-WinSCP_release_5.1.5	1.0	[False, True]
3	2020-05-04	15:16:08	193.105.134.45	SSH-2.0-WinSCP_release_5.7.5	1.0	[False, True]
4	2020-05-04	15:16:19	103.89.89.242	SSH-2.0-paramiko_2.7.1	2.0	[True, False, True]
5	None	None	None	None	NaN	None
6	2020-05-04	15:19:23	193.105.134.45	SSH-2.0-OpenSSH_5.2	1.0	[False, True]
7	2020-05-04	15:22:59	195.3.147.47	SSH-2.0-WinSCP_release_5.1.5	1.0	[False, True]
8	2020-05-04	15:24:47	195.3.147.47	SSH-2.0-OpenSSH_3.9p1	1.0	[False, True]
9	2020-05-04	15:27:16	195.3.147.47	SSH-2.0-paramiko_1.8.1	1.0	[False, True]
10	2020-05-04	15:27:34	195.3.147.47	SSH-2.0-paramiko_1.15.1	1.0	[False, True]
11	2020-05-04	15:28:18	193.105.134.45	SSH-2.0-OpenSSH_4.3	1.0	[False, True]
12	2020-05-04	15:31:25	195.3.147.47	SSH-2.0-OpenSSH_6.1	1.0	[False, True]
13	2020-05-04	15:37:25	195.3.147.47	SSH-2.0-PuTTY_Release_0.62	1.0	[False, True]
14	2020-05-04	15:38:25	195.3.147.47	SSH-2.0-paramiko_1.16.1	1.0	[False, True]
15	2020-05-04	15:40:22	193.105.134.45	SSH-2.0-WinSCP_release_4.1.9	1.0	[False, True]
16	2020-05-04	15:43:05	195.3.147.47	SSH-2.0-paramiko_2.0.2	1.0	[False, True]
17	2020-05-04	15:44:13	109.236.91.85	SSH-2.0-paramiko_2.0.0	1.0	[False, True]
18	2020-05-04	15:44:40	195.3.147.47	SSH-2.0-paramiko_1.16.0	1.0	[False, True]
19	2020-05-04	15:48:33	193.105.134.45	SSH-2.0-OpenSSH_5.3	1.0	[False, True]
20	None	None	None	None	NaN	None
21	2020-05-04	15:53:29	193.105.134.45	SSH-2.0-paramiko_1.10.1	1.0	[False, True]
22	2020-05-04	15:56:31	195.3.147.47	SSH-2.0-WinSCP_release_5.1.3	1.0	[False, True]
23	2020-05-04	15:56:41	193.105.134.45	SSH-2.0-libssh-0.3.4	1.0	[False, True]

	date	time	ip_address	client_type	attempts	password_used
24	2020-05-04	15:57:53	195.3.147.47	SSH-2.0-paramiko_1.7.5	1.0	[False, True]
25	2020-05-04	16:02:04	195.3.147.47	SSH-2.0-paramiko_2.0.2	1.0	[False, True]
26	2020-05-04	16:03:16	195.3.147.47	SSH-2.0-paramiko_1.8.1	1.0	[False, True]
27	2020-05-04	16:03:57	193.105.134.45	SSH-2.0-paramiko_1.15.2	1.0	[False, True]
28	2020-05-04	16:04:00	109.236.91.85	SSH-2.0-JSCH_0.1.51	1.0	[False, True]
29	2020-05-04	16:11:05	109.236.91.85	SSH-2.0-JSCH_0.1.48	1.0	[False, True]
...	...	...	...	...	...	...
18876	2020-05-11	02:33:34	195.3.147.47	SSH-2.0-paramiko_1.15.2	1.0	[False, True]
18877	2020-05-11	02:35:28	195.3.147.47	SSH-2.0-OpenSSH_3.9p1	1.0	[False, True]
18878	2020-05-11	02:35:59	193.105.134.45	SSH-2.0-OpenSSH_5.3	1.0	[False, True]
18879	2020-05-11	02:40:58	193.105.134.45	SSH-2.0-paramiko_2.0.2	1.0	[False, True]
18880	2020-05-11	02:42:28	193.105.134.45	SSH-2.0-WinSCP_release_5.2.7	1.0	[False, True]
18881	2020-05-11	02:47:40	193.105.134.45	SSH-2.0-paramiko_1.16.0	1.0	[False, True]
18882	2020-05-11	02:51:26	109.236.91.85	SSH-2.0-OpenSSH_6.2	1.0	[False, True]
18883	2020-05-11	02:53:25	193.105.134.45	SSH-2.0-PuTTY_Release_0.62	1.0	[False, True]
18884	2020-05-11	02:56:50	195.3.147.47	SSH-2.0-WinSCP_release_5.2.7	1.0	[False, True]
18885	2020-05-11	02:59:11	195.3.147.47	SSH-2.0-PuTTY_Release_0.61	1.0	[False, True]
18886	2020-05-11	03:00:56	195.3.147.47	SSH-2.0-paramiko_1.8.1	2.0	[False, False, True, True]
18887	None	None	None	None	NaN	None
18888	2020-05-11	03:05:48	14.248.209.96	SSH-2.0-Go	1.0	[False, True]
18889	2020-05-11	03:10:07	193.105.134.45	SSH-2.0-OpenSSH_5.3	1.0	[False, True]
18890	2020-05-11	03:10:19	109.236.91.85	SSH-2.0-PuTTY_Release_0.66	1.0	[False, True]
18891	2020-05-11	03:10:46	109.236.91.85	SSH-2.0-libssh-0.3.4	1.0	[False, True]
18892	2020-05-11	03:15:36	193.105.134.45	SSH-2.0-OpenSSH_5.3	1.0	[False, True]

	date	time	ip_address	client_type	attempts	password_used
18893	2020-05-11	03:17:38	195.3.147.47	SSH-2.0-PuTTY_Release_0.63	1.0	[False, True]
18894	2020-05-11	03:17:50	195.3.147.47	SSH-2.0-PuTTY_Release_0.62	1.0	[False, True]
18895	2020-05-11	03:19:20	193.105.134.45	SSH-2.0-WinSCP_release_5.1.3	1.0	[False, True]
18896	2020-05-11	03:21:13	193.105.134.45	SSH-2.0-paramiko_2.1.2	1.0	[False, True]
18897	2020-05-11	03:25:31	195.3.147.47	SSH-2.0-paramiko_1.7.5	1.0	[False, True]
18898	2020-05-11	03:26:56	195.3.147.47	SSH-2.0-WinSCP_release_5.7.4	1.0	[False, True]
18899	2020-05-11	03:27:52	193.105.134.45	SSH-2.0-paramiko_1.16.1	1.0	[False, True]
18900	2020-05-11	03:28:04	193.105.134.45	SSH-2.0-OpenSSH_5.9	1.0	[False, True]
18901	2020-05-11	03:31:31	109.236.81.198	SSH-2.0-sshlib-0.1	1.0	[True]
18902	2020-05-11	03:32:36	195.3.147.47	SSH-2.0-OpenSSH_5.2	1.0	[False, False, True]
18903	None	None	None	None	NaN	None
18904	2020-05-11	03:38:37	195.3.147.47	SSH-2.0-PuTTY_Snapshot_2010_02_20	1.0	[False, True]
18905	2020-05-11	03:38:51	180.183.248.39	SSH-2.0-Go	1.0	[False, True]

18906 rows × 6 columns

Container 103: Both HTTP and HTTPS

In [3]:
```
# make a similar dataframe for log 103

with open('../raw_data/mitm_files/mitm_file103', 'r') as fp:
    # fp is now our file pointer to the mitm data file.

    # skip the five lines at the top
    line = fp.readline()
    line = fp.readline()
    line = fp.readline()
    line = fp.readline()
    line = fp.readline()


    while line:
        curr = analyze_attack(fp)
        if curr is None:
            break
        else:
            mitm_data_103.append(curr)

mitm_df_103 = pd.DataFrame(mitm_data_103, columns = ['date', 'time',
'ip_address', 'client_type', 'attempts', 'password_used' ])
mitm_df_103
```

Out[3]:

	date	time	ip_address	client_type	attempts	password_used
0	2020-05-04	15:29:40	193.105.134.45	SSH-2.0-Granados-1.0	1.0	[False, True]
1	2020-05-04	15:45:47	195.3.147.47	SSH-2.0-paramiko_1.7.7.1	1.0	[False, True]
2	2020-05-04	15:56:02	180.214.238.55	SSH-2.0-Go	1.0	[False, True]
3	2020-05-04	16:01:08	195.3.147.47	SSH-2.0-OpenSSH_5.2	1.0	[False, True]
4	2020-05-04	16:08:14	193.105.134.45	SSH-2.0-libssh_0.5.5	1.0	[False, True]
5	2020-05-04	16:31:13	195.3.147.47	SSH-2.0-OpenSSH_6.2	1.0	[False, True]
6	2020-05-04	16:45:28	195.3.147.47	SSH-2.0-libssh_0.5.5	1.0	[False, True]
7	2020-05-04	16:53:34	195.3.147.47	SSH-2.0-OpenSSH_5.9	1.0	[False, True]
8	2020-05-04	16:55:27	193.105.134.45	SSH-2.0-PuTTY_Release_0.64	1.0	[False, True]
9	2020-05-04	17:12:14	195.3.147.47	SSH-2.0-Nmap_SSH2_Hostkey	1.0	[False, True]
10	2020-05-04	17:13:13	193.105.134.45	SSH-2.0-WinSCP_release_4.1.9	1.0	[False, True]
11	2020-05-04	17:42:53	195.3.147.47	SSH-2.0-libssh_0.5.5	1.0	[False, True]
12	2020-05-04	17:59:36	195.3.147.47	SSH-2.0-PuTTY_Release_0.64	1.0	[False, True]
13	2020-05-04	18:03:32	109.236.91.85	SSH-2.0-paramiko_1.7.5	1.0	[False, True]
14	2020-05-04	18:06:09	193.105.134.45	SSH-2.0-paramiko_1.16.0	1.0	[False, True]
15	2020-05-04	18:16:31	193.105.134.45	SSH-2.0-libssh_0.5.5	1.0	[False, True]
16	2020-05-04	18:36:22	195.3.147.47	SSH-2.0-OpenSSH_6.0	1.0	[False, True]
17	2020-05-04	18:41:08	195.3.147.47	SSH-2.0-paramiko_2.0.2	1.0	[False, True]
18	2020-05-04	18:46:45	141.98.81.83	SSH-2.0-OpenSSH_7.3	1.0	[True]
19	2020-05-04	18:46:47	141.98.81.84	SSH-2.0-OpenSSH_7.3	1.0	[True]
20	2020-05-04	18:46:52	141.98.81.99	SSH-2.0-OpenSSH_7.3	1.0	[True]
21	2020-05-04	18:46:56	141.98.81.107	SSH-2.0-OpenSSH_7.3	1.0	[True]
22	2020-05-04	18:47:00	141.98.81.108	SSH-2.0-OpenSSH_7.3	1.0	[True]

	date	time	ip_address	client_type	attempts	password_used
23	2020-05-04	18:47:04	141.98.81.81	SSH-2.0-OpenSSH_7.3	1.0	[True]
24	2020-05-04	18:47:08	141.98.81.83	SSH-2.0-OpenSSH_7.3	1.0	[True]
25	2020-05-04	18:47:12	141.98.81.84	SSH-2.0-OpenSSH_7.3	1.0	[True]
26	2020-05-04	18:47:15	141.98.81.99	SSH-2.0-OpenSSH_7.3	1.0	[True]
27	2020-05-04	18:47:18	141.98.81.107	SSH-2.0-OpenSSH_7.3	1.0	[True]
28	2020-05-04	18:47:22	141.98.81.108	SSH-2.0-OpenSSH_7.3	1.0	[True]
29	2020-05-04	18:47:26	141.98.81.81	SSH-2.0-OpenSSH_7.3	1.0	[True]
...	...	...	...	...	...	...
15705	2020-05-11	02:42:54	5.101.151.82	SSH-2.0-libssh2_1.4.3	1.0	[True]
15706	2020-05-11	02:43:09	5.101.151.82	SSH-2.0-libssh2_1.4.3	1.0	[True]
15707	2020-05-11	02:43:24	5.101.151.82	SSH-2.0-libssh2_1.4.3	1.0	[True]
15708	2020-05-11	02:43:36	5.101.151.82	SSH-2.0-libssh2_1.4.3	1.0	[True]
15709	2020-05-11	02:43:48	5.101.151.82	SSH-2.0-libssh2_1.4.3	1.0	[True]
15710	2020-05-11	02:44:00	5.101.151.82	SSH-2.0-libssh2_1.4.3	1.0	[True]
15711	2020-05-11	02:44:13	5.101.151.82	SSH-2.0-libssh2_1.4.3	1.0	[True]
15712	2020-05-11	02:44:24	5.101.151.82	SSH-2.0-libssh2_1.4.3	1.0	[True]
15713	2020-05-11	02:44:36	5.101.151.82	SSH-2.0-libssh2_1.4.3	1.0	[True]
15714	2020-05-11	02:44:47	5.101.151.82	SSH-2.0-libssh2_1.4.3	1.0	[True]
15715	2020-05-11	02:44:59	5.101.151.82	SSH-2.0-libssh2_1.4.3	1.0	[True]
15716	2020-05-11	02:45:11	5.101.151.82	SSH-2.0-libssh2_1.4.3	1.0	[True]
15717	2020-05-11	02:45:22	5.101.151.82	SSH-2.0-libssh2_1.4.3	1.0	[True]
15718	2020-05-11	02:45:34	5.101.151.82	SSH-2.0-libssh2_1.4.3	1.0	[True]
15719	2020-05-11	02:45:45	5.101.151.82	SSH-2.0-libssh2_1.4.3	1.0	[True]
15720	2020-05-11	02:45:57	5.101.151.82	SSH-2.0-libssh2_1.4.3	1.0	[True]

	date	time	ip_address	client_type	attempts	password_used
15721	2020-05-11	02:46:08	5.101.151.82	SSH-2.0-libssh2_1.4.3	1.0	[True]
15722	2020-05-11	02:51:31	103.145.253.87	SSH-2.0-paramiko_2.7.1	2.0	[True, False, True]
15723	2020-05-11	02:53:55	193.105.134.45	SSH-2.0-PuTTY_KiTTY	1.0	[False, True]
15724	2020-05-11	03:11:19	195.3.147.47	SSH-2.0-OpenSSH_6.0	1.0	[False, True]
15725	2020-05-11	03:13:15	109.236.91.85	SSH-2.0-WinSCP_release_4.3.5	1.0	[False, True]
15726	2020-05-11	03:19:26	195.3.147.47	SSH-2.0-WinSCP_release_5.7.4	1.0	[False, True]
15727	None	None	None	None	NaN	None
15728	2020-05-11	03:33:57	193.105.134.45	SSH-2.0-PuTTY_Release_0.67	1.0	[False, True]
15729	2020-05-11	03:35:33	193.105.134.45	SSH-2.0-PuTTY_Release_0.62	1.0	[False, True]
15730	2020-05-11	03:35:52	193.105.134.45	SSH-2.0-paramiko_2.0.0	1.0	[False, True]
15731	2020-05-11	03:37:11	109.236.91.85	SSH-2.0-PuTTY_Release_0.62	1.0	[False, True]
15732	2020-05-11	03:37:14	195.3.147.47	SSH-2.0-paramiko_2.0.2	1.0	[False, True]
15733	2020-05-11	03:37:29	195.3.147.47	SSH-2.0-PuTTY_Release_0.62	1.0	[False, True]
15734	2020-05-11	03:38:51	180.183.248.39	SSH-2.0-Go	1.0	[False, True]

15735 rows × 6 columns

Container 101 and Container 104

Unfortunately, there was an error in collecting log data from container 101 that prevented the log file from tracking any data from attempts on the container. As such, we have no data to observe for this particular test case.

Interestingly enough, container 104 had such an erratic Connect and Disconnect pattern that separate events were nigh-indistinguishable. Data from it was not easily convertible into a format similar to containers 102 and 103.

Although not quantitatively definable at this time, this is an interesting observation to make regardless. We can already see this as a clear difference between containers 102, 103, and 104.

Main Dataset: Login attempts

We will now look to collect the login attempts for all of our containers, and their details. This can be found in the `raw_data/logins` folder. Since the data was incorruptible for all of these logs, we will be focusing on this for our main, overall data analysis.

Like before, we'll read the data into lists of lists, then proceed to represent them as dataframes.

Container 101: HTTP

In [4]:
```
# read and scrape from file for container 101

raw_101_logins = []

with open('../raw_data/logins/101.txt', 'r') as fp:

    line = fp.readline()
    while line:
        line_data = re.search("([0-9]{4}-[0-9]{2}-[0-9]{2})\s([0-9]
{2}:[0-9]{2}:[0-9]{2})\.[0-9]*;(.*);password;(.*);(.*)", line)

        if line_data:
            my_date = line_data.group(1)
            my_time = line_data.group(2)
            my_ip = line_data.group(3)
            my_user = line_data.group(4)
            my_password = line_data.group(5)

            raw_101_logins.append([my_date, my_time, my_ip, my_user,
my_password])

        else:
            print("---")
            print("error parsing an event.")
            print("event -> ", line)
            print("---")

        line = fp.readline()

login_df_101 = pd.DataFrame(raw_101_logins, columns = ['date', 'time'
, 'ip_address', 'username', 'password'])
login_df_101
```

Out[4]:

	date	time	ip_address	username	password
0	2020-05-04	15:16:53	141.98.81.150	admin	admin
1	2020-05-04	15:42:58	141.98.81.138	admin	admin
2	2020-05-04	16:12:52	141.98.81.138	admin	admin
3	2020-05-04	16:45:20	141.98.81.138	admin	admin
4	2020-05-04	17:03:07	141.98.81.138	admin	admin
5	2020-05-04	17:24:40	141.98.81.83	root	admin
6	2020-05-04	17:24:44	141.98.81.84	admin	admin1
7	2020-05-04	17:24:48	141.98.81.99	Administrator	admin
8	2020-05-04	17:24:52	141.98.81.107	root	123456
9	2020-05-04	17:24:58	141.98.81.108	admin	ubnt
10	2020-05-04	17:25:02	141.98.81.81	1234	1234
11	2020-05-04	17:25:06	141.98.81.83	guest	guest
12	2020-05-04	17:25:09	141.98.81.84	Admin	Admin
13	2020-05-04	17:25:11	141.98.81.99	root	password
14	2020-05-04	17:25:15	141.98.81.107	admin	123456
15	2020-05-04	17:25:18	141.98.81.108	admin	1234
16	2020-05-04	17:25:22	141.98.81.81	user	1234
17	2020-05-04	18:00:17	141.98.81.138	admin	admin
18	2020-05-04	18:07:23	141.98.81.138	admin	admin
19	2020-05-04	18:14:49	141.98.9.157	admin	admin
20	2020-05-04	18:14:54	141.98.9.159	admin	
21	2020-05-04	18:14:57	141.98.9.160	user	user
22	2020-05-04	18:15:03	141.98.9.161	admin	password
23	2020-05-04	18:15:06	141.98.9.156	root	1234
24	2020-05-04	18:15:10	141.98.9.137	operator	operator
25	2020-05-04	18:15:14	141.98.9.157	test	test
26	2020-05-04	18:15:18	141.98.9.159	root	root
27	2020-05-04	18:15:22	141.98.9.160	guest	guest
28	2020-05-04	18:15:26	141.98.9.161	ubnt	ubnt
29	2020-05-04	18:15:30	141.98.9.156	guest	
...	...	...	...	...	...
3914	2020-05-11	05:25:44	45.136.108.85	user	user
3915	2020-05-11	05:25:45	45.136.108.85	user	user123
3916	2020-05-11	05:25:46	45.136.108.85	user	1234
3917	2020-05-11	05:25:47	45.136.108.85	user	12345

	date	time	ip_address	username	password
3918	2020-05-11	05:25:48	45.136.108.85	user	123456
3919	2020-05-11	05:25:49	45.136.108.85	user	admin
3920	2020-05-11	05:25:55	45.136.108.85	user	baseball
3921	2020-05-11	05:25:57	45.136.108.85	user	1234567890
3922	2020-05-11	05:25:58	45.136.108.85	user	uucp
3923	2020-05-11	05:26:00	45.136.108.85	user	
3924	2020-05-11	05:26:01	45.136.108.85	user	password
3925	2020-05-11	05:26:02	45.136.108.85	user	welcome
3926	2020-05-11	05:26:08	45.136.108.85	user	123
3927	2020-05-11	05:26:15	45.136.108.85	User	
3928	2020-05-11	05:26:21	45.136.108.85	user	111111
3929	2020-05-11	05:26:23	45.136.108.85	user	987654321
3930	2020-05-11	05:26:23	45.136.108.85	user	password123
3931	2020-05-11	05:26:30	45.136.108.85	User	Password
3932	2020-05-11	05:26:36	45.136.108.85	user	123321
3933	2020-05-11	05:26:38	45.136.108.85	user	qwerty
3934	2020-05-11	05:26:39	45.136.108.85	user	test
3935	2020-05-11	05:26:39	45.136.108.85	user	ubnt
3936	2020-05-11	05:26:40	45.136.108.85	user	nopass
3937	2020-05-11	05:26:41	45.136.108.85	user	football
3938	2020-05-11	05:26:46	45.136.108.85	user	public
3939	2020-05-11	05:26:53	45.136.108.85	user1	
3940	2020-05-11	05:26:54	45.136.108.85	user1	nopass
3941	2020-05-11	05:26:54	45.136.108.85	user1	user1
3942	2020-05-11	05:26:55	45.136.108.85	user1	1234
3943	2020-05-11	05:27:00	141.98.81.150	admin	admin

3944 rows × 5 columns

Container 102: HTTPS

In [5]:
```
# read and scrape from file for container 102

raw_102_logins = []

with open('../raw_data/logins/102.txt', 'r') as fp:

    line = fp.readline()
    while line:
        line_data = re.search("([0-9]{4}-[0-9]{2}-[0-9]{2})\s([0-9]
{2}:[0-9]{2}:[0-9]{2})\.[0-9]*;(.*);password;(.*);(.*)", line)

        if line_data:
            my_date = line_data.group(1)
            my_time = line_data.group(2)
            my_ip = line_data.group(3)
            my_user = line_data.group(4)
            my_password = line_data.group(5)

            raw_102_logins.append([my_date, my_time, my_ip, my_user,
my_password])

        else:
            print("---")
            print("error parsing an event.")
            print("event -> ", line)
            print("---")

        line = fp.readline()

login_df_102 = pd.DataFrame(raw_102_logins, columns = ['date', 'time'
, 'ip_address', 'username', 'password'])
login_df_102
```

Out[5]:

	date	time	ip_address	username	password
0	2020-05-04	15:14:02	195.3.147.47	admin	<Any pass>
1	2020-05-04	15:14:09	195.3.147.47	22	admin
2	2020-05-04	15:14:22	195.3.147.47	admin	12345
3	2020-05-04	15:16:10	193.105.134.45	admin	letmein
4	2020-05-04	15:16:20	103.89.89.242	root	root
5	2020-05-04	15:18:19	193.105.134.45	admin	<No Pass>
6	2020-05-04	15:18:19	193.105.134.45	anonymous	<No pass>
7	2020-05-04	15:19:25	193.105.134.45	admin	admin
8	2020-05-04	15:23:01	195.3.147.47	123321	111111
9	2020-05-04	15:24:49	195.3.147.47	admin	letmein
10	2020-05-04	15:27:17	195.3.147.47	testuser	ftp
11	2020-05-04	15:27:36	195.3.147.47	test	123456
12	2020-05-04	15:28:20	193.105.134.45	123	123
13	2020-05-04	15:31:27	195.3.147.47	support	helpdesk
14	2020-05-04	15:37:27	195.3.147.47	root	12345678
15	2020-05-04	15:38:27	195.3.147.47	ubnt	client
16	2020-05-04	15:40:24	193.105.134.45	default	<No pass>
17	2020-05-04	15:43:06	195.3.147.47	ubnt	shadow
18	2020-05-04	15:44:15	109.236.91.85	ubnt	admin
19	2020-05-04	15:44:41	195.3.147.47	test	159753
20	2020-05-04	15:48:35	193.105.134.45	test	<No Pass>
21	2020-05-04	15:48:36	195.3.147.47	anonymous	<No pass>
22	2020-05-04	15:53:30	193.105.134.45	root	root
23	2020-05-04	15:56:32	195.3.147.47	22	backup
24	2020-05-04	15:56:43	193.105.134.45	sshd	service
25	2020-05-04	15:57:55	195.3.147.47	22	ubnt
26	2020-05-04	16:02:05	195.3.147.47	admin	password
27	2020-05-04	16:03:18	195.3.147.47	1111	1111
28	2020-05-04	16:03:59	193.105.134.45	root	12345678
29	2020-05-04	16:04:02	109.236.91.85	test	123123
...	...	...	...	...	...
20196	2020-05-11	04:45:43	195.3.147.47	sshd	sshd
20197	2020-05-11	04:48:00	109.236.91.85	root	root
20198	2020-05-11	04:48:09	221.238.161.66	guest	666666
20199	2020-05-11	04:49:11	195.3.147.47	test	123123

	date	time	ip_address	username	password
20200	2020-05-11	04:53:17	195.3.147.47	demo	demo
20201	2020-05-11	04:53:19	195.3.147.47	ubnt	ubnt
20202	2020-05-11	04:53:52	195.3.147.47	administrator	admin123
20203	2020-05-11	04:58:27	109.236.81.198	root	root
20204	2020-05-11	04:59:07	195.3.147.47	test	159753
20205	2020-05-11	05:02:05	193.105.134.45	administrator	1qaz2wsx
20206	2020-05-11	05:03:55	195.3.147.47	101	101
20207	2020-05-11	05:08:10	193.105.134.45	123	letmein
20208	2020-05-11	05:10:44	195.3.147.47	backup	backup
20209	2020-05-11	05:11:00	193.105.134.45	admin	password
20210	2020-05-11	05:11:39	109.236.91.85	administrator	1qaz2wsx
20211	2020-05-11	05:12:25	193.105.134.45	ubnt	client
20212	2020-05-11	05:12:25	195.3.147.47	default	<No pass>
20213	2020-05-11	05:13:54	195.3.147.47	test	<No Pass>
20214	2020-05-11	05:14:12	193.105.134.45	test	123123
20215	2020-05-11	05:14:51	171.238.159.249	ubnt	
20216	2020-05-11	05:14:55	193.105.134.45	root	12345678
20217	2020-05-11	05:15:35	193.105.134.45	ubnt	admin
20218	2020-05-11	05:19:26	195.3.147.47	123	123
20219	2020-05-11	05:20:11	195.3.147.47	admin	password
20220	2020-05-11	05:20:26	195.3.147.47	anonymous	<No pass>
20221	2020-05-11	05:21:43	195.3.147.47	ubnt	shadow
20222	2020-05-11	05:22:34	195.3.147.47	admin	admin
20223	2020-05-11	05:23:37	195.3.147.47	666666	666666
20224	2020-05-11	05:24:09	195.3.147.47	admin	<Any pass>
20225	2020-05-11	05:25:56	193.105.134.45	support	helpdesk

20226 rows × 5 columns

Container 103: Both HTTP and HTTPS

In [6]:
```
# read and scrape from file for container 103

raw_103_logins = []

with open('../raw_data/logins/103.txt', 'r') as fp:

    line = fp.readline()
    while line:
        line_data = re.search("([0-9]{4}-[0-9]{2}-[0-9]{2})\s([0-9]
{2}:[0-9]{2}:[0-9]{2})\.[0-9]*;(.*);password;(.*);(.*)", line)

        if line_data:
            my_date = line_data.group(1)
            my_time = line_data.group(2)
            my_ip = line_data.group(3)
            my_user = line_data.group(4)
            my_password = line_data.group(5)

            raw_103_logins.append([my_date, my_time, my_ip, my_user,
my_password])

        else:
            # omitting debug code here to yield presentable results
            pass # omit erroneous log data (this file contains some)

        line = fp.readline()

login_df_103 = pd.DataFrame(raw_103_logins, columns = ['date', 'time'
, 'ip_address', 'username', 'password'])
login_df_103
```

Out[6]:

	date	time	ip_address	username	password
0	2020-05-04	15:29:41	193.105.134.45	111111	1234
1	2020-05-04	15:45:49	195.3.147.47	3comcso	RIP000
2	2020-05-04	15:56:04	180.214.238.55	root	root
3	2020-05-04	16:01:09	195.3.147.47	123321	11111
4	2020-05-04	16:08:16	193.105.134.45	root	root
5	2020-05-04	16:31:15	195.3.147.47	root	alpine
6	2020-05-04	16:45:30	195.3.147.47	123321	11111
7	2020-05-04	16:53:36	195.3.147.47	111111	1234
8	2020-05-04	16:55:28	193.105.134.45	123321	11111
9	2020-05-04	17:12:16	195.3.147.47	111111	111111
10	2020-05-04	17:13:15	193.105.134.45	root	alpine
11	2020-05-04	17:42:55	195.3.147.47	111111	admin
12	2020-05-04	17:59:38	195.3.147.47	adfexc	adfexc
13	2020-05-04	18:03:34	109.236.91.85	adfexc	adfexc
14	2020-05-04	18:06:11	193.105.134.45	111111	111111
15	2020-05-04	18:16:32	193.105.134.45	root	root
16	2020-05-04	18:36:24	195.3.147.47	root	root
17	2020-05-04	18:41:09	195.3.147.47	3comcso	RIP000
18	2020-05-04	18:46:47	141.98.81.83	root	admin
19	2020-05-04	18:46:50	141.98.81.84	admin	admin1
20	2020-05-04	18:46:53	141.98.81.99	Administrator	admin
21	2020-05-04	18:46:58	141.98.81.107	root	123456
22	2020-05-04	18:47:03	141.98.81.108	admin	ubnt
23	2020-05-04	18:47:06	141.98.81.81	1234	1234
24	2020-05-04	18:47:10	141.98.81.83	guest	guest
25	2020-05-04	18:47:14	141.98.81.84	Admin	Admin
26	2020-05-04	18:47:17	141.98.81.99	root	password
27	2020-05-04	18:47:20	141.98.81.107	admin	123456
28	2020-05-04	18:47:25	141.98.81.108	admin	1234
29	2020-05-04	18:47:28	141.98.81.81	user	1234
...	...	...	...	...	...
17036	2020-05-11	02:45:47	5.101.151.82	test1	test1
17037	2020-05-11	02:45:58	5.101.151.82	user1	user1
17038	2020-05-11	02:46:10	5.101.151.82	user01	user01
17039	2020-05-11	02:51:33	103.145.253.87	root	root

	date	time	ip_address	username	password
17040	2020-05-11	02:52:06	193.105.134.45	111111	1234
17041	2020-05-11	02:53:56	193.105.134.45	3comcso	RIP000
17042	2020-05-11	03:11:20	195.3.147.47	123321	11111
17043	2020-05-11	03:13:17	109.236.91.85	root	root
17044	2020-05-11	03:19:28	195.3.147.47	adfexc	adfexc
17045	2020-05-11	03:33:07	193.105.134.45	adfexc	adfexc
17046	2020-05-11	03:33:59	193.105.134.45	111111	admin
17047	2020-05-11	03:35:35	193.105.134.45	root	root
17048	2020-05-11	03:35:53	193.105.134.45	111111	111111
17049	2020-05-11	03:37:13	109.236.91.85	root	alpine
17050	2020-05-11	03:37:16	195.3.147.47	root	alpine
17051	2020-05-11	03:37:31	195.3.147.47	root	root
17052	2020-05-11	03:38:53	180.183.248.39	supervisor	klv123
17053	2020-05-11	03:41:56	195.3.147.47	111111	admin
17054	2020-05-11	03:49:46	195.3.147.47	3comcso	RIP000
17055	2020-05-11	04:06:24	180.252.85.164	user	admin
17056	2020-05-11	04:11:47	195.3.147.47	111111	111111
17057	2020-05-11	04:16:12	193.105.134.45	123321	11111
17058	2020-05-11	04:48:08	221.238.161.66	guest	666666
17059	2020-05-11	04:49:43	109.236.91.85	3comcso	RIP000
17060	2020-05-11	04:54:35	109.236.91.85	111111	1234
17061	2020-05-11	05:06:08	185.200.250.156	admin	admin
17062	2020-05-11	05:06:12	14.186.226.79	admin	password
17063	2020-05-11	05:10:36	193.105.134.45	111111	admin
17064	2020-05-11	05:14:51	171.238.159.249	ubnt	
17065	2020-05-11	05:26:17	195.3.147.47	111111	1234

17066 rows × 5 columns

Container 104: Control

In [7]:
```
# read and scrape from file for container 104

raw_104_logins = []

with open('../raw_data/logins/104.txt', 'r') as fp:

    line = fp.readline()
    while line:
        line_data = re.search("([0-9]{4}-[0-9]{2}-[0-9]{2})\s([0-9]
{2}:[0-9]{2}:[0-9]{2})\.[0-9]*;(.*);password;(.*);(.*)", line)

        if line_data:
            my_date = line_data.group(1)
            my_time = line_data.group(2)
            my_ip = line_data.group(3)
            my_user = line_data.group(4)
            my_password = line_data.group(5)

            raw_104_logins.append([my_date, my_time, my_ip, my_user,
my_password])

        else:
            print("---")
            print("error parsing an event.")
            print("event -> ", line)
            print("---")

        line = fp.readline()

login_df_104 = pd.DataFrame(raw_104_logins, columns = ['date', 'time'
, 'ip_address', 'username', 'password'])
login_df_104
```

Out[7]:

	date	time	ip_address	username	password
0	2020-05-04	15:27:29	193.105.134.45	test	test
1	2020-05-04	15:29:34	193.105.134.45	admin	guest
2	2020-05-04	15:33:41	193.105.134.45	1234	111111
3	2020-05-04	15:35:46	195.3.147.47	admin	guest
4	2020-05-04	15:47:47	193.105.134.45	admin	ubnt
5	2020-05-04	15:48:00	180.214.238.55	root	root
6	2020-05-04	15:48:18	193.105.134.45	123321	root
7	2020-05-04	16:11:46	193.105.134.45	111111	1234
8	2020-05-04	16:15:11	193.105.134.45	1502	1502
9	2020-05-04	16:33:03	195.3.147.47	admin	guest
10	2020-05-04	16:34:56	129.2.19.206	LIBRSYSISVC	fbsn-WVNdHuaE4uS!GT7mEJa
11	2020-05-04	16:35:00	129.2.19.206	sach	D$$C@r3s
12	2020-05-04	16:35:04	129.2.19.206	public	
13	2020-05-04	16:35:08	129.2.19.206	AD\uenwesi	15Oldsapleroad
14	2020-05-04	16:43:47	193.105.134.45	0	0
15	2020-05-04	17:01:29	188.255.113.206	root	root
16	2020-05-04	17:01:30	188.255.113.206	root	admin
17	2020-05-04	17:01:30	188.255.113.206	root	12345
18	2020-05-04	17:01:30	188.255.113.206	root	guest
19	2020-05-04	17:01:31	188.255.113.206	root	123456
20	2020-05-04	17:01:31	188.255.113.206	root	1234
21	2020-05-04	17:01:36	188.255.113.206	root	123
22	2020-05-04	17:01:36	188.255.113.206	root	hlL0mlNAabiR
23	2020-05-04	17:01:37	188.255.113.206	root	test
24	2020-05-04	17:01:37	188.255.113.206	root	toor
25	2020-05-04	17:01:37	188.255.113.206	root	qwerty
26	2020-05-04	17:01:37	188.255.113.206	root	pfsense
27	2020-05-04	17:01:41	188.255.113.206	root	J5cmmu=Kyf0-br8CsW
28	2020-05-04	17:01:44	188.255.113.206	admin	admin
29	2020-05-04	17:01:44	188.255.113.206	admin	123456
...	...	...	...	...	...
5375	2020-05-11	03:42:29	5.101.151.83	spark	spark
5376	2020-05-11	03:42:37	5.101.151.83	db	db
5377	2020-05-11	03:42:43	5.101.151.83	server	server
5378	2020-05-11	03:42:50	5.101.151.83	dev	dev

	date	time	ip_address	username	password
5379	2020-05-11	03:42:57	5.101.151.83	webadmin	webadmin
5380	2020-05-11	03:43:04	5.101.151.83	ftpuser	ftpuser
5381	2020-05-11	03:43:11	5.101.151.83	ftpadmin	ftpadmin
5382	2020-05-11	03:43:18	5.101.151.83	ftptest	ftptest
5383	2020-05-11	03:43:25	5.101.151.83	debian	debian
5384	2020-05-11	03:43:32	5.101.151.83	test1	test1
5385	2020-05-11	03:43:38	5.101.151.83	user1	user1
5386	2020-05-11	03:43:45	5.101.151.83	user01	user01
5387	2020-05-11	03:46:56	195.3.147.47	123	123
5388	2020-05-11	03:47:12	193.105.134.45	1234	111111
5389	2020-05-11	03:52:59	195.3.147.47	test	test
5390	2020-05-11	03:56:15	193.105.134.45	root	root
5391	2020-05-11	04:06:24	180.252.85.164	user	admin
5392	2020-05-11	04:09:56	193.105.134.45	111111	1234
5393	2020-05-11	04:19:38	195.3.147.47	111111	1234
5394	2020-05-11	04:47:47	180.214.238.55	root	root
5395	2020-05-11	04:48:08	221.238.161.66	guest	666666
5396	2020-05-11	04:49:52	195.3.147.47	0	0
5397	2020-05-11	05:07:47	193.105.134.45	admin	ubnt
5398	2020-05-11	05:08:32	193.105.134.45	12345	12345
5399	2020-05-11	05:11:41	193.105.134.45	test	test
5400	2020-05-11	05:13:48	193.105.134.45	1502	1502
5401	2020-05-11	05:14:52	171.238.159.249	ubnt	
5402	2020-05-11	05:15:33	193.105.134.45	111111	1234
5403	2020-05-11	05:20:58	193.105.134.45	123321	root
5404	2020-05-11	05:22:25	193.105.134.45	111111	111111

5405 rows × 5 columns

Section 2: Tidying Data

In this case, the data we were provided is highly unique in that it is fairly clean. It is organized by date and time, and is already sorted. However, there are still small adjustments we wish to make.

Main Dataset

Identifying Use of No Password

I noticed when performing my exploratory data analysis that there were multiple cases when no password was used when attempting to login to our machines. (In other words, the password parameter was left empty'.) However, this was treated differently by our MITM software. We could see results that said 'no pass' or 'any pass' within angle brackets, along with totally blank fields. An easily noticeable example is that of container 102's password field.

In [8]:
```
login_df_102['password']
```

Out[8]:

0	<Any pass>
1	admin
2	12345
3	letmein
4	root
5	<No Pass>
6	<No pass>
7	admin
8	111111
9	letmein
10	ftp
11	123456
12	123
13	helpdesk
14	12345678
15	client
16	<No pass>
17	shadow
18	admin
19	159753
20	<No Pass>
21	<No pass>
22	root
23	backup
24	service
25	ubnt
26	password
27	1111
28	12345678
29	123123
...	...
20196	sshd
20197	root
20198	666666
20199	123123
20200	demo
20201	ubnt
20202	admin123
20203	root
20204	159753
20205	1qaz2wsx
20206	101
20207	letmein
20208	backup
20209	password
20210	1qaz2wsx
20211	client
20212	<No pass>
20213	<No Pass>
20214	123123
20215	
20216	12345678
20217	admin
20218	123
20219	password
20220	<No pass>
20221	shadow

```
20222         admin
20223        666666
20224    <Any pass>
20225      helpdesk
Name: password, Length: 20226, dtype: object
```

I'm opting to convert all of these variants to 'None' objects, so we can identify which attempts were passwordless with Python later. I employ regex here to make the operation easier.

In [9]:
```
def identify_no_password(input_string):
    if not input_string:
        return None
    nopass = re.search('^<no pass>$', input_string, re.IGNORECASE)
    anypass = re.search('^<any pass>$', input_string, re.IGNORECASE)
    if nopass or anypass:
        return None
    else:
        return input_string

login_df_102['password'] = login_df_102['password'].apply(identify_no
_password)
login_df_102['password']
```

Out[9]:

0	None
1	admin
2	12345
3	letmein
4	root
5	None
6	None
7	admin
8	111111
9	letmein
10	ftp
11	123456
12	123
13	helpdesk
14	12345678
15	client
16	None
17	shadow
18	admin
19	159753
20	None
21	None
22	root
23	backup
24	service
25	ubnt
26	password
27	1111
28	12345678
29	123123
	...
20196	sshd
20197	root
20198	666666
20199	123123
20200	demo
20201	ubnt
20202	admin123
20203	root
20204	159753
20205	1qaz2wsx
20206	101
20207	letmein
20208	backup
20209	password
20210	1qaz2wsx
20211	client
20212	None
20213	None
20214	123123
20215	None
20216	12345678
20217	admin
20218	123
20219	password
20220	None
20221	shadow

```
20222       admin
20223      666666
20224        None
20225    helpdesk
Name: password, Length: 20226, dtype: object
```

Now, we might as well apply the transformation to the other login attempt datframes.

In [10]:
```
login_df_101['password'] = login_df_101['password'].apply(identify_no
_password)
login_df_103['password'] = login_df_103['password'].apply(identify_no
_password)
login_df_104['password'] = login_df_104['password'].apply(identify_no
_password)
```

Supplementary Dataset

Removing Empty Entries

One artifact of my data analysis script is that, when an attack isn't open and closed in one occurrence, the event is discarded. (This was done because it was highly difficult to separate these data points into separate events). These results produce empty rows in our dataframe to signify a 'broken event'. I wish to do two things with this measurement. I'm looking to store the number of blank entries in each dataframe (for later data analysis), then remove these blank rows.

In this case, we will be measuring how many 'None' rows exist in the 102 and 103 containers

In [11]:
```
count_102 = mitm_df_102.count()
count_103 = mitm_df_103.count()

none_count_102 = len(mitm_df_102) - count_102[0]
none_count_103 = len(mitm_df_103) - count_103[0]
print("'None' counts for containers 102 and 103")
print(none_count_102)
print(none_count_103)
```

```
'None' counts for containers 102 and 103
887
1011
```

Section 3: Interpretation of Results

Drawing conclusions from the data we've come up with. Since we only have one week of data, it's a little hard to draw definitive conclusions, but here I've included meaningful visualizations of the data we've produced.

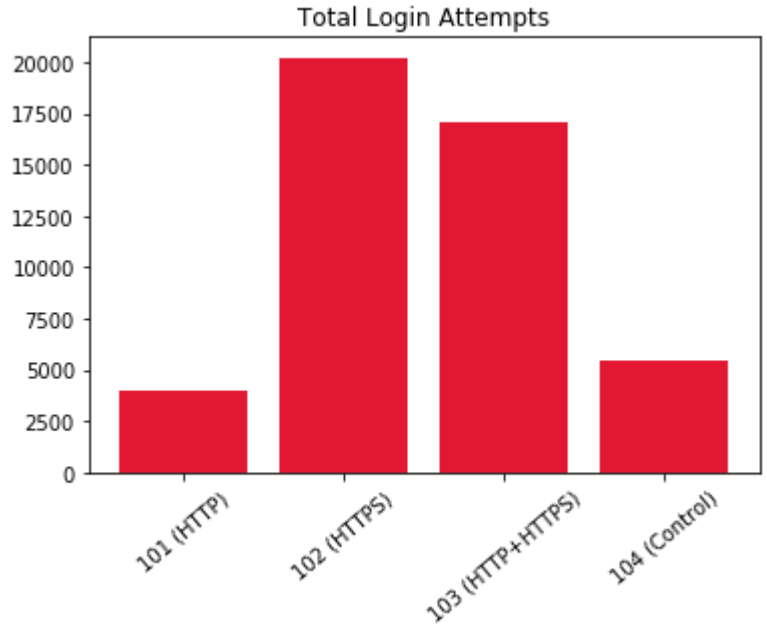
General Data: Overview

Main dataset: Login Attempts

Looking at our main dataset's login attempts, here are some useful statistics.

In [12]:
```
# total overall login attempts
bar = {u'101 (HTTP)':len(login_df_101), u'102 (HTTPS)':len(login_df_1
02), u'103 (HTTP+HTTPS)':len(login_df_103), u'104 (Control)':len(logi
n_df_104),}

plt.bar(*zip(*bar.items()),color='#E21833')
plt.xticks(rotation='40')
plt.title("Total Login Attempts")
plt.savefig('../plots/total_attempts.png')
plt.show()
```

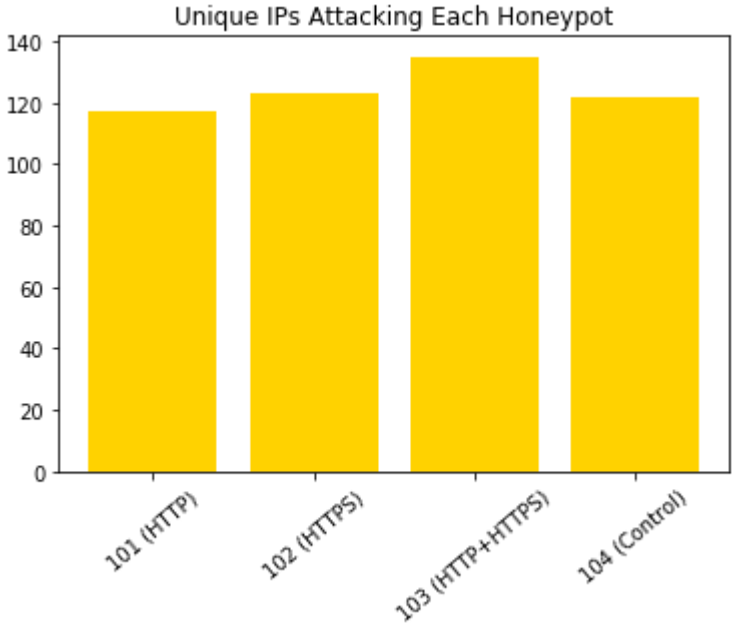


We can make two inferences given the above data. First of all, we can see that the presence of HTTPS in a honeypot makes it immensely more attractive in terms of login attempts. It seems that most modern attackers target systems with HTTPS capability. This could potentially be due to the potential for more lucrative data to be stored on HTTPS-enabled machines vs. HTTP enabled machines.

In [13]:
```
# unique attacker ips per honeypot
unique_ips_101 = len(login_df_101['ip_address'].unique())
unique_ips_102 = len(login_df_102['ip_address'].unique())
unique_ips_103 = len(login_df_103['ip_address'].unique())
unique_ips_104 = len(login_df_104['ip_address'].unique())

bar = {u'101 (HTTP)':unique_ips_101, u'102 (HTTPS)':unique_ips_102, u
'103 (HTTP+HTTPS)':unique_ips_103, u'104 (Control)':unique_ips_104,}

plt.bar(*zip(*bar.items()),color='#ffd200')
plt.xticks(rotation='40')
plt.title("Unique IPs Attacking Each Honeypot")
plt.savefig('../plots/unique_ips_count.png')
plt.show()
```



Interestingly, we can see that roughly the same amount of different hosts attacked all of our honeypots equally. This makes the previous graph doubly interesting- it's not that the presence of HTTPS attracted more distinct hosts attacking our machine, it's that the presence of HTTPS may have encouraged attackers to make multiple attempts to compromise our system.

Username and Password Frequency

In [14]:
```
# usernames and popularity per honeypot
unique_user_101 = (login_df_101['username'].unique())
unique_user_102 = (login_df_102['username'].unique())
unique_user_103 = (login_df_103['username'].unique())
unique_user_104 = (login_df_104['username'].unique())

login_df_101['username'].value_counts(dropna=False).rename_axis('uniq
ue usernames').to_frame('counts')
```

Out[14]:

unique usernames	counts
root	1066
admin	844
user	160
test	117
ubnt	105
guest	91
ubuntu	79
oracle	55
support	55
postgres	41
operator	36
	35
adm	33
pi	33
nagios	31
ftp	31
1234	30
super	30
ftpuser	29
administrator	28
default	27
git	22
odoo	20
student	20
Administrator	19
manager	18
user1	17
centos	17
jenkins	16
111111	15
...	...
0101	1
64.34.161.5	1
64.225.113.64	1

unique usernames	counts
139.196.224.52	1
47.102.219.96	1
193.207.252.104	1
138.68.12.89	1
888888	1
sniffer	1
666666	1
guset	1
csftp	1
zimbra	1
95.142.160.67	1
107.149.115.110	1
kali	1
47.190.71.40	1
149.56.249.91	1
3.120.219.45	1
tes	1
142.93.126.73	1
35.156.0.76	1
ethos	1
181.236.183.127	1
197.230.222.34	1
156.96.12.34	1
172.120.205.74	1
csgo	1
67.222.62.53	1
161.117.228.56	1

246 rows × 1 columns

In [15]:
```
login_df_102['username'].value_counts(dropna=False).rename_axis('uniq
ue usernames').to_frame('counts')
```

Out[15]:

unique usernames	counts
root	13290
admin	1270
test	557
ubnt	523
22	439
default	243
administrator	239
123	217
sshd	217
support	176
user	165
testuser	128
demo	121
anonymous	114
backup	113
123321	111
0	110
666666	109
1111	109
101	108
guest	104
ubuntu	94
oracle	79
	52
ftpuser	51
ftp	43
nagios	39
operator	38
postgres	34
adm	33
...	...
drcom	1
droplet	1
unbt	1

unique usernames	counts
ts	1
testing	1
dropbox	1
root;1qaz0p	1
user2	1
vps	1
appadmin	1
debug	1
dspace	1
qwer	1
drcomadmin	1
indra	1
rufus	1
devuser	1
data	1
syslog	1
musicbot	1
alfresco	1
www-data	1
phpmy	1
baikal	1
telegraf	1
user3	1
httpfs	1
http	1
kafka	1
app	1

321 rows × 1 columns

In [16]:
```
login_df_103['username'].value_counts(dropna=False).rename_axis('uniq
ue usernames').to_frame('counts')
```

Out[16]:

unique usernames	counts
root	13460
admin	643
111111	331
user	161
test	135
3comcso	113
123321	112
adfexc	108
ubuntu	104
guest	103
ubnt	96
oracle	74
support	66
	51
ftp	46
ftpuser	45
operator	36
nagios	34
adm	33
manager	33
pi	31
super	30
1234	30
administrator	28
default	28
postgres	28
user1	23
student	21
testuser	21
jenkins	20
...	...
phpmy	1
db2inst1	1
root;Password.	1

unique usernames	counts
root;9ol.0p	1
nvidia	1
indra	1
orangepi	1
deluge	1
guset	1
csserver	1
rig	1
webuser	1
root;Qaz123	1
spam	1
arkserver	1
devuser	1
erp	1
root;asdfghjkl	1
linaro	1
fake	1
root;Qwerty#0p	1
game	1
sdtdserver	1
kali	1
clamav	1
httpd	1
vnc	1
xmr	1
odroid	1
sinusbot	1

285 rows × 1 columns

In [17]:
```
login_df_104['username'].value_counts(dropna=False).rename_axis('uniq
ue usernames').to_frame('counts')
```

Out[17]:

unique usernames	counts
root	1366
admin	849
test	229
111111	228
user	163
1234	137
1502	112
123321	112
0	110
12345	110
123	108
ubnt	104
guest	95
ubuntu	65
oracle	64
support	58
	51
ftp	44
ftpuser	44
operator	38
adm	33
nagios	31
postgres	31
manager	31
super	30
administrator	28
default	28
user1	23
git	23
test1	20
...	...
checkfsys	1
mediator	1
duni	1

unique usernames	counts
orangepi	1
install	1
m1122	1
tour	1
MCVEADMIN	1
accounting	1
tutor	1
config	1
lpadm	1
lynx	1
mountsys	1
halt	1
PCUSER	1
HELPDESK	1
file	1
WP	1
admina	1
0101	1
xmr	1
odroid	1
distech	1
xbian	1
IntraSwitch	1
bananapi	1
rig	1
airlines	1
NETPRIV	1

278 rows × 1 columns

Overall, we can make a few observations from this username data.

- First, we can say that the most common usernames are **root** and **admin**. This is to be expected, and doesn't change between all 4 of our honeypots
- Other common attempts appear to be **user** and **test** which were used with varying frequency across the systems
- An interesting surprise is that 111111 is not common between 101 and 102, but appears in 103 and 104. This could say something about the configuration of the system the attacker(s) attempting 111111.
- Various IP addresses were used as usernames when logins were made on the HTTP-only honeypot. This could mean that less sophisticated crawlers could be specially attacking HTTP-only systems, given that our other honeypots did not pick up this input

Aside from those observations, we can see that there is no other largely noticeable difference. Overall, it seems that for the majority of attackers, username was not a concern when differentiating between honeypots with HTTP, HTTPS, and HTTP+HTTPS capability.

In [18]:
```
# password variation
pw_df_101 = login_df_101['password'].value_counts(dropna=False).renam
e_axis('unique passwords').to_frame('counts')
pw_df_101
```

Out[18]:

unique passwords	counts
admin	471
123456	158
1234	136
root	119
NaN	116
password	112
ubnt	89
123	77
test	72
user	70
12345	66
12345678	60
123456789	58
1234567890	56
guest	46
ubuntu	41
p@ssw0rd	33
P@ssw0rd	32
111111	31
987654321	30
support	30
raspberry	29
operator	27
oracle	26
root123	26
1	25
postgres	24
test123	24
0	22
default	21
...	...
hacked	1
ubuntuubuntu	1
smcadmin	1

unique passwords	counts
1011	1
testfree	1
deploy	1
45.249.78.113	1
nagios123	1
176.31.230.54	1
useruser	1
hatori	1
testing	1
arjunapambudi1	1
server123	1
bruteme123!	1
210.118.227.82	1
148.251.155.50	1
nagiosnagios	1
121.36.205.15	1
Athira2011	1
linux	1
hadoop123	1
180.215.44.71	1
167.71.39.21	1
118.89.194.72	1
54.228.233.24	1
lwefcs4kGgvJDQ	1
nginx	1
5h0n9kh0	1
git123	1

453 rows × 1 columns

In [19]:
```
pw_df_102 = login_df_102['password'].value_counts(dropna=False).renam
e_axis('unique passwords').to_frame('counts')
pw_df_102
```

Out[19]:

unique passwords	counts
NaN	666
admin	501
root	379
password	356
ubnt	290
123456	289
12345	229
backup	221
letmein	218
123	199
12345678	162
1234	144
111111	138
0	134
admin123	131
ftp	130
client	128
demo	126
master	126
1qaz2wsx	123
123123	119
shadow	117
service	116
159753	115
1111	115
666666	111
sshd	111
helpdesk	111
888888	108
101	108
...	...
Pa44w0rd	1
584131421	1
Admin@	1

unique passwords	counts
p@SSw0RD	1
1@3$5^7*9)	1
arbeitg	1
yzidc0524g	1
needidc	1
mickey	1
1P@ssw0rd	1
336699a	1
3edc	1
lkjhgfg	1
654321a.	1
buxingag	1
Qwert@123456	1
sunwayit	1
!@#123wsx	1
112	1
Letmein@123456	1
P@$$w0rd2121	1
1.0	1
210684	1
admin_!@#	1
p@ssw0rd8	1
adm1n1234	1
4444444444g	1
center	1
web123456g	1
!@#123QWAS	1

8873 rows × 1 columns

In [20]:
```
pw_df_103 = login_df_103['password'].value_counts(dropna=False).renam
e_axis('unique passwords').to_frame('counts')
pw_df_103
```

Out[20]:

unique passwords	counts
admin	293
root	267
1234	251
123456	189
password	142
NaN	133
111111	130
11111	119
RIP000	113
alpine	112
12345	111
adfexc	108
test	98
123	93
ubnt	81
user	75
guest	65
12345678	58
123456789	54
1234567890	51
ubuntu	44
default	42
P@ssw0rd	39
support	37
p@ssw0rd	33
987654321	30
super	28
raspberry	27
operator	27
1	26
...	...
9000idc	1
2003g	1
!@@QW	1

unique passwords	counts
662766	1
xiaohuai	1
!qazxsw23e	1
Pa$$w0rd555	1
P@ssw0rd741	1
112	1
!password	1
Qwerasdf1234	1
2014	1
HUlHUl	1
QWASZX	1
sistema	1
Adm1n	1
pakistang	1
zaq1_2wsx	1
123@qweasd@	1
zzidc@123	1
q123456789	1
ADm1n123	1
138888	1
1123581321	1
QWEqwe123!	1
By123456	1
!QAZ9ijn	1
)(*&^%$#@!	1
QWER	1
mt@123	1

9141 rows × 1 columns

In [21]:
```
pw_df_104 = login_df_104['password'].value_counts(dropna=False).renam
e_axis('unique passwords').to_frame('counts')
pw_df_104
```

Out[21]:

unique passwords	counts
root	454
1234	248
111111	238
12345	222
test	202
ubnt	193
123	191
admin	185
123456	170
guest	168
0	136
password	135
NaN	120
1502	112
user	72
12345678	59
1234567890	58
123456789	58
ubuntu	38
P@ssw0rd	38
default	38
p@ssw0rd	37
oracle	30
987654321	30
super	28
support	27
operator	27
test123	26
qwerty	24
1	23
...	...
P@ssword	1
accounting	1
indra	1

unique passwords	counts
oracle12345	1
nagios123	1
23091992	1
nasoture	1
fal	1
lineprin	1
PDP11	1
XC4rrAT6KEGr	1
manda011185	1
umountfsys	1
SYSTEM	1
nginx	1
temppwd	1
REMOTE	1
Password1	1
xxxx	1
system_admin	1
web	1
unix	1
tour	1
nagiosnagios	1
m1122	1
danger	1
hadoop123	1
install	1
sync	1
rig	1

498 rows × 1 columns

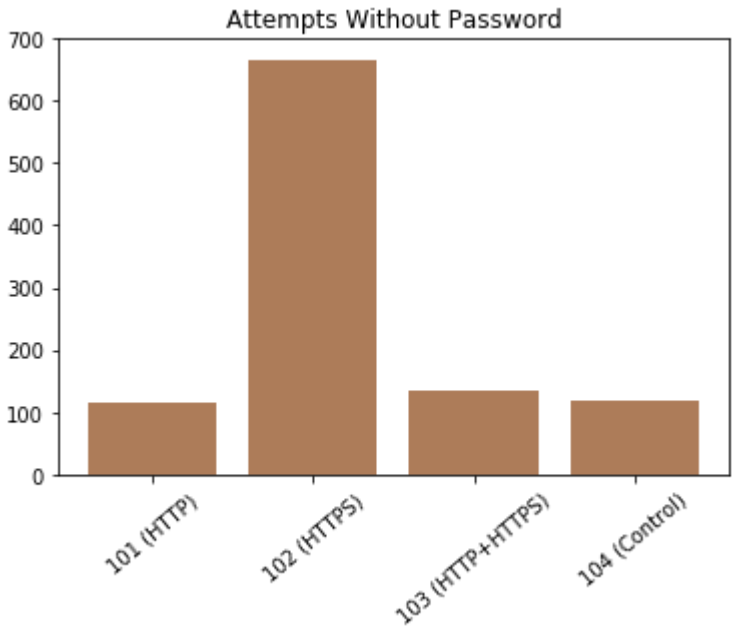
Right off the bat, we can see a much higher variation in the passwords selected by attackers. It's pretty clear that they are producing more creative suggestions than their usernames, which could arguably be due to the following philosophy: **Many users will change the default password after setting up a machine, but significantly less of those users will change the superuser account/bother setting up a new one.**

However, to examine a fairly unique pattern that involves the lack of a password, let's examine how exactly attempts with and without a password stacked up for our 4 honeypots.

In [22]:
```
# attempts with and without password
no_pw_count_101 = len(login_df_101) - login_df_101.count()[4]
no_pw_count_102 = len(login_df_102) - login_df_102.count()[4]
no_pw_count_103 = len(login_df_103) - login_df_103.count()[4]
no_pw_count_104 = len(login_df_104) - login_df_104.count()[4]

bar = {u'101 (HTTP)':no_pw_count_101, u'102 (HTTPS)':no_pw_count_102,
u'103 (HTTP+HTTPS)':no_pw_count_103, u'104 (Control)':no_pw_count_104
,}

plt.bar(*zip(*bar.items()),color='#AD7C59')
plt.xticks(rotation='40')
plt.title("Attempts Without Password")
plt.savefig('../plots/no_pw_attempts.png')
plt.show()
```



We can see in this graph that surprisingly, the amount of attempts without using a password is surprisingly disproportionate. The difference between 101 (HTTP) and 102 (HTTPS) in terms of passwordless attempts could be chalked up to the sheer difference in total attempts, but seeing that 103 (HTTP+HTTPS) also has a similar amount of attempted attacks on it, we can find that for some reason, **Only HTTPS** attracts attackers with more passwordless attempts.

Miscellaneous: Potential Interesting Takeaways

General Password Selection - Oddities

We found some odd passwords in use. Aside from the expected defaults such as "admin", "root", "123456789", etc., we also detected attackers trying to login with passwords such as "baseball", "football", and "letmein". We didn't expect such oddities to appear in brute force attempts, and once can only surmise that the inclusion of these attempts could imply that various insecure systems in the past shared these seemingly arbitrary passwords.

We also saw seemingly randomly generated passwords, such as 'D$$C@r3s', which also seemed equally pointless. However, some user and password combinations seemed oddly specific, such as the following. *user:* LIBRSYSISVC, *pass:* fbsn-WVNdHuaE4uS!GT7mEJa. It leads us to believe that these attackers could be exploiting known, albeit niche, vulnerabilities established on smaller, lesser known systems. (E.g. LIBRSYSISVC, although it yielded a fairly fruitless google search, could be the username on a lesser-known type of system that the attacker is specifically looking for).

We also found a bunch of nonsense phrases, like 'csgo' and 'fake'. These, we can perhaps chalk up to throwaway attempts by attackers.

'None' counts vs. Total Attacks

Analyzing the 'None' counts for our containers vs. the total attack counts yields an interesting result.

In [23]:
```
print("'None' counts for containers 102 and 103 vs. Total attack counts:")
print(none_count_102, len(mitm_df_102))
print(none_count_103, len(mitm_df_103))
print()

print("Ratios for 102 (HTTPS) vs 103 (HTTP and HTTPS)")
print(none_count_102/len(mitm_df_102))
print(none_count_103/len(mitm_df_103))
```

```
'None' counts for containers 102 and 103 vs. Total attack counts:
887 18906
1011 15735

Ratios for 102 (HTTPS) vs 103 (HTTP and HTTPS)
0.046916322860467576
0.06425166825548141
```

Overall, we see a 4.69% 'chaos factor' in the HTTPS honeypot, and a 6.43% 'chaos factor' in the HTTP+HTTPS honeypot. Although this difference is not substantial, perhaps the erratic nature of the attacks was increased minutely due to the extra presence of the HTTP port open.