Security - Continued

Fall 2018 – Introduction to Full-Stack Development

Instructor: Matthew Fritter

matt@frit.me

Stepping Back: Lab 4

- In Lab 4, we focused on looking at implementing a couple security measures on our website:
  - Stopping server information leakage and camouflaging our server
  - Stopping XSS attacks using a client-side Content Security Policy
  - A brief review of server-side XSS protection and SQL injection
- Recall that XSS and SQL injection account for >50% of all vulnerabilities/exploits at the present time

Review From Last Lecture: XSS

- Cross-Site Scripting (XSS) is a means of executing arbitrary client-side script (usually JavaScript), for malicious or disruptive purposes
- Two primary types, Persistent and Reflected:
  - Persistent XSS attacks are stored server-side in a database field or file that is then echoed out to users, executing the code in question
  - Reflected XSS attacks have the payload script inserted into a GET query parameter or similar tag-along request data field
- Extremely prominent security flaw that poses a serious risk to our users

Review From Last Lecture: SQLi

- SQL Injection (SQLi) is similar to XSS in that it relies on injection of code
- Rather than injecting a script, the attacker injects arbitrary SQL into a query string via an un-sanitized user input
- Various methods of SQL injection that allow for modifying of an existing query or execution of a completely different query or update
- Like XSS, extremely high-risk in web applications

Review From Last Lecture: Protecting Your Server

- First and foremost, remember the golden rule: ***Never trust user-generated content!***
  - This means sanitizing all your inputs for possible script or SQL before it has a chance to be echoed out to users or a query is run
  - To help defend against XSS attacks, enable client-side XSS protection and implement a Content Security Policy to limit the ability to execute off-site resources
  - Combination of client-side browser protection and server-side sanitization and validation of data provides good protection against XSS attacks

Announcements

- There is no quiz today – rescheduled to October 23rd (2 weeks from now)
    - Quiz will be an interactive Capture-The-Flag security exercise using a test server I will provide you during the quiz
    - Can be done in teams of up to 4
    - Marks will be awarded for each flag found – first team to find all flags will receive a prize
- No class or lab next week – October 16th.

Questions

- Are there any remaining questions about Lab 4?
- Are there any questions about the upcoming (October 23rd) quiz?

Security – Information Leakage

★ Information Leakage

- We discussed this a bit in last lecture and a bit in the lab
- Information leakage is the unintended release of potentially sensitive data
  - This could be information regarding your server environment and web application architecture
  - It could be user-specific data (!)
- Common areas of information leakage:
  - Server Headers
  - Error Pages
  - Code comments
  - Misconfigured APIs

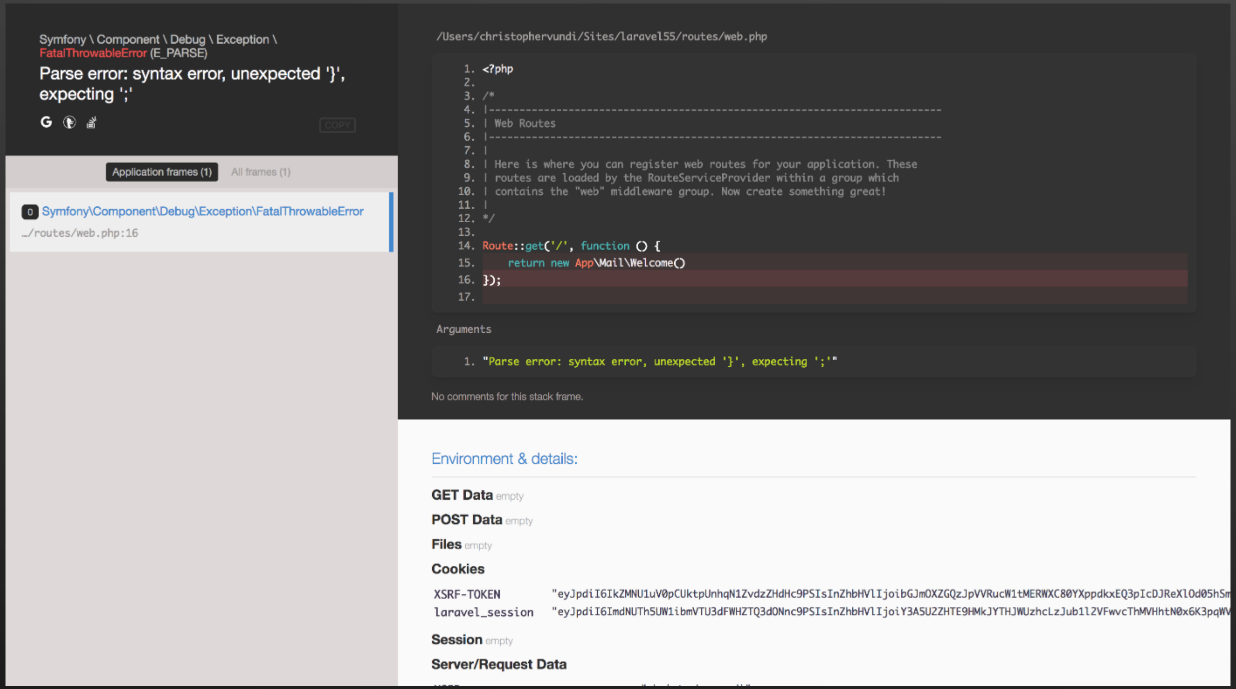
★ Information Leakage Dangers

- Recall from last week: *Obscurity is not Security*
- Simply preventing information leakage and relying on that to protect our server from attacks isn’t a valid security strategy
- However, it also makes little sense to provide attackers more potential information about our systems
  - The more information they have about your environment, your web server, your back-end architecture, your database – the easier it will be to pinpoint vulnerabilities in these components that could be an attack vector
  - If any of your components have zero-day exploits, you could be allowing attackers to quickly identify your server as vulnerable, making it a target

Where is Your Site Leaking Information?

- Think – Where is your server leaking information? Hint: (Almost) all of your servers will be leaking data due to this feature.
- I brought up this specific leak issue way back in Lecture 2

Where is Your Site Leaking Information?



Symfony \ Component \ Debug \ Exception \
FatalThrowableError (E_PARSE)

Parse error: syntax error, unexpected '}', expecting ';'

Symfony\Component\Debug\Exception\FatalThrowableError
…/routes/web.php:16

/Users/christophervundi/Sites/laravel55/routes/web.php

```
<?php

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

Route::get('/', function () {
    return new App\Mail\Welcome()
});

```

Arguments

```
1. "Parse error: syntax error, unexpected '}', expecting ';'"
```

No comments for this stack frame.

Environment & details:

GET Data empty

POST Data empty

Files empty

Cookies

```
XSRF-TOKEN       "eyJpdiI6IkZMNU1uV0pCUktpUnhqN1ZvdzZHdHc9PSIsInZhbHVlIjoibGJmOXZGQzJpVVRucW1tMERWXC80YXppdkxEQ3pIcDJReXlOd05hSm
laravel_session  "eyJpdiI6ImdNUTh5UW1ibmVTU3dFWHZTQ3dONnc9PSIsInZhbHVlIjoiY3A5U2ZHTE9HMkJYTHJWUzhcLzJub1l2VFwvcThMVHhtN0x6K3pqWV
```

Session empty

Server/Request Data

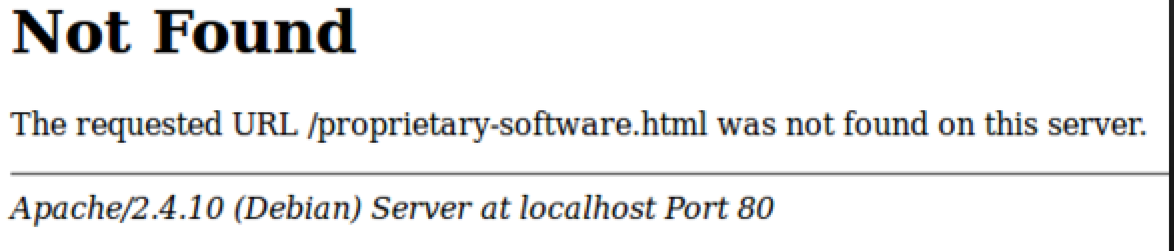
- The Laravel stack trace error pages are a perfect example of information leakage. Potentially valuable information could be mined from this page by purposefully trying to create back-end errors

Another Typical Information Leak

- Remember from Lab 4, we added the following line to our Apache configuration file:

```
ServerSignature Off
```

- The server signature on Apache default error pages is a common information leak, especially if you don't expect Apache error pages to be displayed



Not Found

The requested URL /proprietary-software.html was not found on this server.

Apache/2.4.10 (Debian) Server at localhost Port 80

Less-Thought of Information Leaks

- Consider the typical email password reset system that is used by many websites that have user accounts
- Have you ever noticed that many of them will *not* tell you if you've entered a valid email address or username? They just say the email has been sent *if* that email or username exists
  - Reasoning behind that: If such a system informed the user they'd entered an invalid username, an attacker could use the system to "fish" for usernames and emails of users
- However, many sites don't perform the same action in sign-up pages, allowing someone to potentially fish for usernames and emails there (assuming they must be unique)

★ Dealing with Information Leaks

- Many information leaks are part of default configurations, and simply need to be disabled (consider Apache headers and signatures, PHP X-Powered-By, etc)
- Other than that, you just have to think carefully: how could this system be abused? What is the most damaging information that someone with a great deal of time (or a great deal of machines) could pull out of this feature?
- It is almost impossible to completely eliminate information leakage. There is always some degree of metadata associated with everything we return to the user
    - Trying to minimize the most common ways that environment data is mined is a good idea, however

Security – Camouflage

The Idea behind Camouflage

- Going beyond stopping information leakage, server camouflage is the purposeful leakage of false data in an attempt to misdirect attackers
    - We did this in Lab 4 when we setup our Apache servers to identify as Microsoft IIS servers
    - The point? Attacks aimed at IIS servers, for example, will for the most part completely fail against Apache or NGINX servers due to architectural differences
    - Depending on the level of obfuscation, can provide some degree of relief from automatic vulnerability scanners (although don’t count on it)

★ Camouflage and its Role

- Camouflage, like preventing information leakage in the first place, isn't *really* a security measure – remember, Obscurity is not Security
  - What we're doing here is we're reducing the signal-to-noise ratio. By warding off "dumb" or strictly-bound bots with false information, we can focus on more "real" threats
  - Plenty of bots out there, particularly those used by opportunists (script kiddies) will implicitly trust Server and X-Powered-By HTTP headers, because they assume that many (even most) web servers leave them in default configurations

★ Fingerprinting Webservers

- While changing the HTTP headers to appear like a different web server or architecture may fool bots and users that simply trust the Server HTTP header, There are many ways of identifying a server that don’t rely on this
- *Fingerprinting* is the technique of identifying underlying software (often the web server, sometimes the operating system) via responses
  - Common discrepancies checked:
    - Ordering of HTTP header fields
    - Looking for default pages (info.php, phpmyadmin, etc)
    - Checking server response to various errors (incorrect HTTP version, unsupported HTTP request types, etc)

Fingerprinting Webservers, Continued

- Using large databases of existing server signatures, software such as HTTPrint and HTTPRecon can fingerprint web servers – even camouflaged ones – fairly reliably
  - In Lab 4, we managed to fool HTTPRecon – but HTTPrint (ironically, an older tool) still managed to identify our server as an Apache server (albeit the wrong version)
  - While things like header ordering could be manually changed to help throw off fingerprinting attempts, it is for the most part more effort than it is worth
- As I mentioned previously, it's almost impossible to completely eliminate information leakage – there will always be *something* that can identify your server if someone looks hard enough

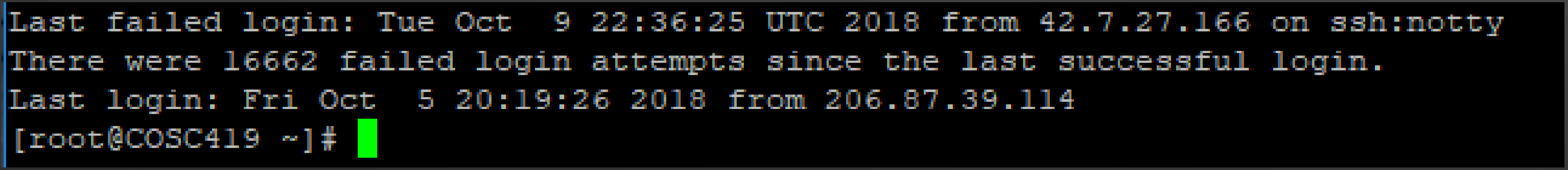
Comparison of Security Headers

```
HTTP/1.1 200 OK
Date: Sun, 15 Jun 2003 17:10: 49 GMT
Server: Apache/1.3.23
Last-Modified: Thu, 27 Feb 2003 03:48: 19 GMT
ETag: 32417-c4-3e5d8a83
Accept-Ranges: bytes
Content-Length: 196
Connection: close
Content-Type: text/HTML
```

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Expires: Yours, 17 Jun 2003 01:41: 33 GMT
Date: Mon, 16 Jun 2003 01:41: 33 GMT
Content-Type: text/HTML
Accept-Ranges: bytes
Last-Modified: Wed, 28 May 2003 15:32: 21 GMT
ETag: b0aac0542e25c31: 89d
Content-Length: 7369
```

Security – SSH Login Security

SSH Login Attempts

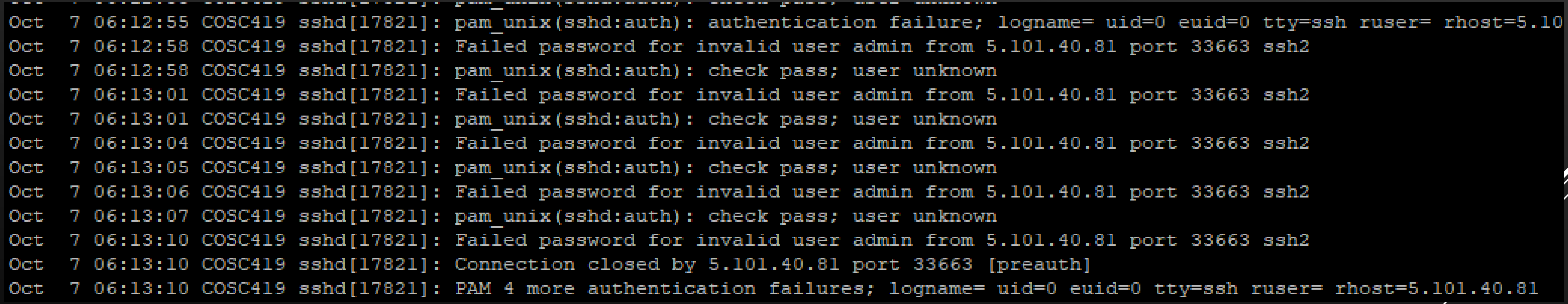


```
Last failed login: Tue Oct  9 22:36:25 UTC 2018 from 42.7.27.166 on ssh:notty
There were 16662 failed login attempts since the last successful login.
Last login: Fri Oct  5 20:19:26 2018 from 206.87.39.114
[root@COSC419 ~]#
```

- That’s an average of 2.83 attempted logins every minute of every day of every week
- DigitalOcean servers all fall within specific IP ranges, and are a particularly common target for people trying to gain access using common logins

Looking At your SSH Log

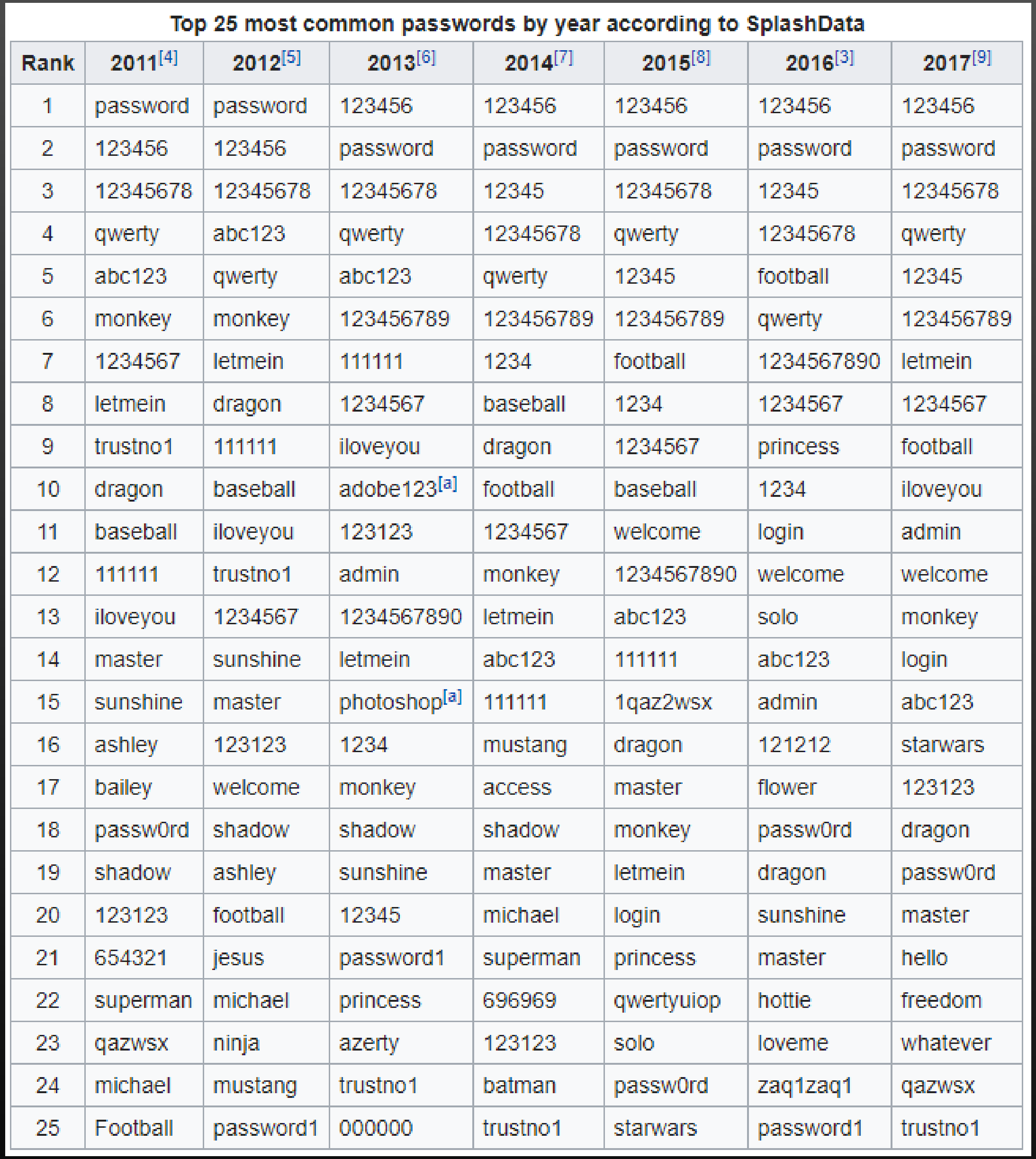
- Your SSH log can be found at /var/log/secure
- This file logs attempted logins, including usernames attempted, number of failures, and originating IP address



```
Oct  7 06:12:55 COSC419 sshd[17821]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=5.10
Oct  7 06:12:58 COSC419 sshd[17821]: Failed password for invalid user admin from 5.101.40.81 port 33663 ssh2
Oct  7 06:12:58 COSC419 sshd[17821]: pam_unix(sshd:auth): check pass; user unknown
Oct  7 06:13:01 COSC419 sshd[17821]: Failed password for invalid user admin from 5.101.40.81 port 33663 ssh2
Oct  7 06:13:01 COSC419 sshd[17821]: pam_unix(sshd:auth): check pass; user unknown
Oct  7 06:13:04 COSC419 sshd[17821]: Failed password for invalid user admin from 5.101.40.81 port 33663 ssh2
Oct  7 06:13:05 COSC419 sshd[17821]: pam_unix(sshd:auth): check pass; user unknown
Oct  7 06:13:06 COSC419 sshd[17821]: Failed password for invalid user admin from 5.101.40.81 port 33663 ssh2
Oct  7 06:13:07 COSC419 sshd[17821]: pam_unix(sshd:auth): check pass; user unknown
Oct  7 06:13:10 COSC419 sshd[17821]: Failed password for invalid user admin from 5.101.40.81 port 33663 ssh2
Oct  7 06:13:10 COSC419 sshd[17821]: Connection closed by 5.101.40.81 port 33663 [preauth]
Oct  7 06:13:10 COSC419 sshd[17821]: PAM 4 more authentication failures; logname= uid=0 euid=0 tty=ssh ruser= rhost=5.101.40.81
```

★ The Brute Force Intrusion

- These attempts to gain access to our server are almost invariably ***brute force attacks***
- Brute force attacks simply rely on being able to make enough guesses (enough time, enough requests, enough machines) that they'll eventually get it right – the thousand monkeys with typewriters problem
- Brute force techniques may use a generator to create permutations of alpha-numeric text, but this is usually inefficient – a list of common passwords (and usernames) is the most prevalent
- This is why password *length* and *uniqueness* are a problem!



Top 25 most common passwords by year according to SplashData

Rank	2011[4]	2012[5]	2013[6]	2014[7]	2015[8]	2016[3]	2017[9]
1	password	password	123456	123456	123456	123456	123456
2	123456	123456	password	password	password	password	password
3	12345678	12345678	12345678	12345	12345678	12345	12345678
4	qwerty	abc123	qwerty	12345678	qwerty	12345678	qwerty
5	abc123	qwerty	abc123	qwerty	12345	football	12345
6	monkey	monkey	123456789	123456789	123456789	qwerty	123456789
7	1234567	letmein	111111	1234	football	1234567890	letmein
8	letmein	dragon	1234567	baseball	1234	1234567	1234567
9	trustno1	111111	iloveyou	dragon	1234567	princess	football
10	dragon	baseball	adobe123[a]	football	baseball	1234	iloveyou
11	baseball	iloveyou	123123	1234567	welcome	login	admin
12	111111	trustno1	admin	monkey	1234567890	welcome	welcome
13	iloveyou	1234567	1234567890	letmein	abc123	solo	monkey
14	master	sunshine	letmein	abc123	111111	abc123	login
15	sunshine	master	photoshop[a]	111111	1qaz2wsx	admin	abc123
16	ashley	123123	1234	mustang	dragon	121212	starwars
17	bailey	welcome	monkey	access	master	flower	123123
18	passw0rd	shadow	shadow	shadow	monkey	passw0rd	dragon
19	shadow	ashley	sunshine	master	letmein	dragon	passw0rd
20	123123	football	12345	michael	login	sunshine	master
21	654321	jesus	password1	superman	princess	master	hello
22	superman	michael	princess	696969	qwertyuiop	hottie	freedom
23	qazwsx	ninja	azerty	123123	solo	loveme	whatever
24	michael	mustang	trustno1	batman	passw0rd	zaq1zaq1	qazwsx
25	Football	password1	000000	trustno1	starwars	password1	trustno1

★ The Brute Force Intrusion

- Short passwords can be easily guessed via permutation
    - Computation power required to guess a password goes up massively in proportion to it’s length – combinations are factorial operations
    - Hence, why most modern password systems apply minimum password length requirements
- Common (non-unique) passwords can be easily guessed from lists
    - “12345678”, “qwerty”, and common phrases from books and movies can often be guessed very quickly, even for long phrases
- Presently, your servers each use a 10-character randomly generated password consisting of upper, lower, and numeric ASCII characters
    - Fairly secure – for now

★ Preventing SSH Intrusion – Protection Techniques

- Option 1: Use a very secure password
- Option 2: Fail2Ban
- Option 3: SSH Keys
- Option 4 (kind of): SSH Port Change

Preventing SSH Intrusion – Protection Techniques

- Option 1: Use a very secure password
  - Pros: Easy, doesn’t require anything new
  - Cons: Have to remember a long (think 32 characters +) password, still have to deal with constant messages about attempted logins
- Option 2: Fail2Ban
- Option 3: SSH Keys
- Option 4 (kind of): SSH Port Change

Preventing SSH Intrusion – Protection Techniques

- Option 1: Use a very secure password
- Option 2: Fail2Ban
  - Pros: Fairly easy to setup and install, works well when configured properly
  - Cons: Can accidentally ban yourself, still leaves some log messages behind
- Option 3: SSH Keys
- Option 4 (kind of): SSH Port Change

Preventing SSH Intrusion – Protection Techniques

- Option 1: Use a very secure password
- Option 2: Fail2Ban
- Option 3: SSH Keys
  - Pros: No more need for passwords
  - Cons: Sharing server access can become difficult, if you lose your SSH key you have to find a way to gain access to the server again
- Option 4 (kind of): SSH Port Change

Preventing SSH Intrusion – Protection Techniques

- Option 1: Use a very secure password
- Option 2: Fail2Ban
- Option 3: SSH Keys
- Option 4 (kind of): SSH Port Change
  - Pros: Simple and eliminates most automated SSH intrusions
  - Cons: Not really a security measure, more obfuscation

★ Fail2Ban

- Fail2Ban is a service that is designed to automatically ban potential malicious actors after a set number of failed login attempts via SSH
- Installing Fail2Ban is easy:
  - Double check that you have `epel-release` installed
  - Then run: `yum install fail2ban`
  - Enable it on startup with: `systemctl enable fail2ban`
- Fail2Ban works by using the IPTables service to block connections from certain IP addresses once it has detected a configurable number of failed login attempts

Fail2Ban Configuration Basics

- Using the following command, you can edit the Fail2Ban configuration file:

```
nano /etc/fail2ban/jail.local
```

  - Note, don’t edit the jail.conf file – if Fail2Ban updates, there is a good chance that file will be overwritten
- Some basic directives in the file:
  - bantime – specifies the amount of time (s) to ban the user
  - findtime – window of time (s) to check for retries
  - maxretry – maximum number of tries before being banned
  - port – allows you to define which port to watch

An Example Fail2Ban Configuration

```
[DEFAULT]
bantime = 86400
findtime = 300
maxretry = 3
banaction = iptables-multiport

[sshd]
enabled = true
port = ssh
```

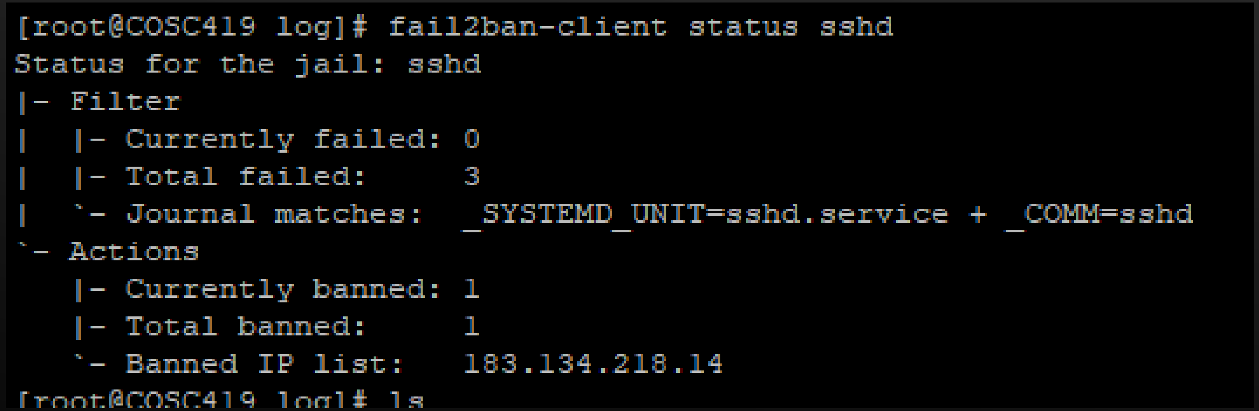
- Translation:
- For all jails:
  - Ban for 24 hours
  - If more than 3 tries made within 5 minutes
  - Use IPTables
- For the SSHD jail:
  - Enable the jail
  - Set the port to SSH (22)

Fail2Ban Configuration, Continued

- Once you’ve set up your configuration file, you’ll need to restart the Fail2Ban service
- You can check the number of currently banned Ips and the number of failed login attempts using the following command:

```
fail2ban-client status sshd
```

  - Note: if you’re using a jail other than the sshd jail, you can specify it instead of sshd to get its status



```
[root@COSC419 log]# fail2ban-client status sshd
Status for the jail: sshd
|- Filter
|  |- Currently failed: 0
|  |- Total failed:     3
|  `- Journal matches:  _SYSTEMD_UNIT=sshd.service + _COMM=sshd
`- Actions
   |- Currently banned: 1
   |- Total banned:     1
   `- Banned IP list:   183.134.218.14
[root@COSC419 log]# ls
```

★ Changing the SSH Port

- As I mentioned, this is mostly an obfuscation measure – a dedicated attacker using a port scanner will easily identify your SSH ports, even if you use a different port number
- However, this does work exceptionally well for deflecting most automated SSH intrusions (which will make up about 99.999% of the intrusions you'll face)
- First, you'll need to edit your sshd configuration file at `/etc/ssh/sshd_config` – look for a line that says Port 22 (probably commented out) – uncomment it and change it to the port of your choice, save and close
  - It is very important you pick a port that isn't in use for another protocol: 80 (HTTP), 443 (SSL), 20/21 (FTP), 23 (Telnet), etc…

★ Changing the SSH Port

- Now we need to update SELinux to let it know that we have changed the default SSH port:

```
semanage port -a -t ssh_port_t -p tcp <port #>
```

- Then, restart sshd:

```
systemctl restart sshd
```

- If you close your terminal and try to re-SSH into your machine, you should receive a connection refused error using port 22; your new port should be active
    - Note: if you screw this up, it *is* possible to lock yourself out of your machine, necessitating a root password reset by the data center

★ Preventing SSH Intrusion

- Preventing SSH intrusion (Password-based, not SSH Keys) is best done using a combination of the aforementioned practices:
  - Use a good password to avoid first-time lucky guesses
  - Use Fail2Ban to ban brute force abusers and bots
  - Use an alternate SSH port to help reduce the amount of bot traffic
    - If you specify the port in your jail.local fail2ban configuration, you can have fail2ban monitor your newly-assigned SSH port

Security – The Root Account

★(Not) Using the Root Account

- Up until now, many of you probably used the root account that you were given for working on your labs and experimenting with your web server
  - This is generally considered very bad practice!
  - The `root` account is omnipotent, and it can be very dangerous in the wrong hands
- It’s better to add a new user and give it **super user** (su) privileges and prevent the root account from being remotely accessed

★ Why is the Root Account Dangerous?

- The root account can edit and change any file in the system without restriction – even if doing so will cause serious damage to the system
- In addition, the root account is a known target, while another username generally isn't (pick a username wisely)
  - If you allow remote login to root, you're doing half the work for an attacker – they no longer have to guess a username (because it's root), they just have to guess a password
  - For this reason, root is usually either disabled entirely, or only allowed to be logged in from the localhost
- On an actual development server, this should usually be one of the first steps you take to help harden your server

Creating a New Account

- First, use the adduser command to add a user with the specified username:

```
adduser <username here>
```

- Next, assign a password for the user account. This will prompt you to enter a password for the account and verify it:

```
Passwd <username here>
```

- Presently, the account only has normal user permissions. To give it Superuser permissions, we need to add it to the special group *wheel*:

```
gpasswd –a <username here> wheel
```

- You should now be able to use *su* to switch to the new user, and *sudo* to perform commands that would require superuser (root) access:

```
su <username here>
```

Disabling Remote Root Login

- Switch back the root account using *su* (it will prompt you for the root password)
- Open the sshd config file:

```
nano /etc/ssh/sshd_config
```

- Find the line that says "#PermitRootLogin yes", change it to no, and uncomment the line (remove the #). Save and close the file.
- Restart the sshd service
- Now, you can try disconnecting and reconnecting – you'll find that you are now unable to login as root, and an error is returned if you try

Closing Material

Lab 5 Preview

- In lab 5, we'll be touching on the various SSH security measures that we've looked at today, in addition to studying the idea of server fingerprinting a little bit more
- We'll also be looking at the Modsecurity package, which is an all-in-one security package. We'll talk about it briefly next lecture, but it's mostly a hands-on activity
- Goal is to be done with security this week, so that we come back in two weeks we can start fresh on the next topic (Wordpress and Flask)

Upcoming Quiz

- October 23rd – Topics from lectures 4 and 5, as well as labs 4 and 5.
- Cooperative team exercise – you pick your own teams (or work on your own, if you want) – up to four people
- Come ready and prepared. You may want to bring some quick and dirty scripts with you to quickly parse webpage information to look for security flaws, but you'll have enough time to find them in class
- A bit of explanation: Capture The Flag (CTF) is common in many hackathons and security competitions. "Flags" containing secret phrases may be placed in files, metadata, anywhere – and contestants must try to find them while circumventing security and access control measures
  - For us, flags will be hidden behind common security flaws that we've discussed so far
  - This may include within web pages/requests (including behind authentication), or on the server itself (accessed via SSH).

That’s All Folks

Next Week – No Class!