Tapestry5HowToMitigateLoginAttacks

How To Mitigate Login Attacks

Brute force and dictionary attacks use recursive attempts to guess passwords. One of the most effective approaches to mitigate this is to implement a delay between login attempts, which effectively slows down these scripts just enough to render them useless. The problem with implementing 'account lockout' as an alternative is that its open for abuse, and also may create some administrative overhead. This short article uses a simplified example to illustrate this principle.

1. The first step is to create a simple Pojo that represents a failed login, we give it the ability to count failed attempts, and give it a time to live.

```
/**
 * @author Peter Stavrinides
 *
 */
public class FailedLogin {

	/** Tracks the number of failed login attempts */
	private int failedLoginCount_ = 0;

	/** The number of times sleep was incurred */
	private int sleepCount = 0;

	/** The delay period (10 minutes) */
	private int SLEEP_INTERVAL = 600000;

	/** The login expired time */
	private Date expireTime_ = null;


	/** 2 Hours of inactivity expires the counter */
	private final int TIME_TO_LIVE = 2;


	/**
	 * A method to increment the failed logins
	 * @throws InterruptedException
	 */
	public void incrementFailedLoginCount() throws InterruptedException {

		//reset all counters if the object expired
		if (isExpired()) {
			sleepCount = 0;
			failedLoginCount_ = 0;
		}

		 //increment the failed login counter
		failedLoginCount_ += 1;

		//every 5th failed login we double the sleep period
		if(failedLoginCount_ == 5){
			sleepCount +=1;		failedLoginCount_ = 0;
			Thread.sleep(SLEEP_INTERVAL);
			SLEEP_INTERVAL = (SLEEP_INTERVAL * 2);
		}

		//set the expires time
		Calendar ts = Calendar.getInstance();
		ts.add(Calendar.HOUR, 2);
		expireTime_ = new Date(ts.getTimeInMillis());
	}

	/** @return true if this object is expired */
	public boolean isExpired() {
		if(expireTime_ == null){
			return false;
		}
		Date now = new Date();
		long diff = now.getTime() - expireTime_.getTime();
		int hours = (int) (Math.floor(diff / 1000 / 60 / 60));
```

```
                // the object has expired
                if (hours >= TIME_TO_LIVE) {
                        return true;
                }
                return false;
        }

        /**
         * @throws InterruptedException
         */
        public FailedLogin() throws InterruptedException{
                incrementFailedLoginCount();
        }

}
```

2. The next step is to create a Tapestry Singleton service. The service uses a map to store failed login attempts, and uses the callers IP address as an identifier.

```
public class FailedLoginTracker {

        /** A map to store the FailedLogin objects */
        private static ConcurrentHashMap<String, FailedLogin> failedLogins_ = new ConcurrentHashMap<String,
FailedLogin>();


        /**
         * @param ipAddress
         * @throws InterruptedException
         */
        public void setFailedLogin(String ipAddress) throws InterruptedException{
                FailedLogin login = null;
                if(failedLogins_.containsKey(ipAddress)){
                        login  = failedLogins_.get(ipAddress);
                        login.incrementFailedLoginCount();
                }else{
                        //construction ensures increment by 1
                        login = new FailedLogin();
                }
                failedLogins_.put(ipAddress, login);

                // Not essential, but I prefer to take out the trash
                cleanUpExpired();
        }


        /**
         * A method to clean the map of failed logins
         * that have passed their time to live
         */
        private void cleanUpExpired() {
                for(String ip : failedLogins_.keySet()) {
                        if(failedLogins_.get(ip).isExpired())
                                failedLogins_.remove(ip);
                }
        }
}
```

3. Lastly we add a simple binding in AppModule... a Singleton is the default scope so we only require the following:

```
binder.bind(FailedLoginTracker.class)
```

Conclusion

I have tried to keep the example above concise, but there are a several enhancements that come to mind:

- The most obvious being more intelligent tracking options other than just by IP, as these can be easily faked.
- It also might be worth checking multiple user names tried by the same IP, or if each failed attempt by the same user / IP uses a different session id

By combining these checks intelligently you can get a good indication of a scripted attack.

... any suggestions, corrections etc. are encouraged!

Peter

Suggestion:
After 3 failed login attempts, show a CAPTCHA image.