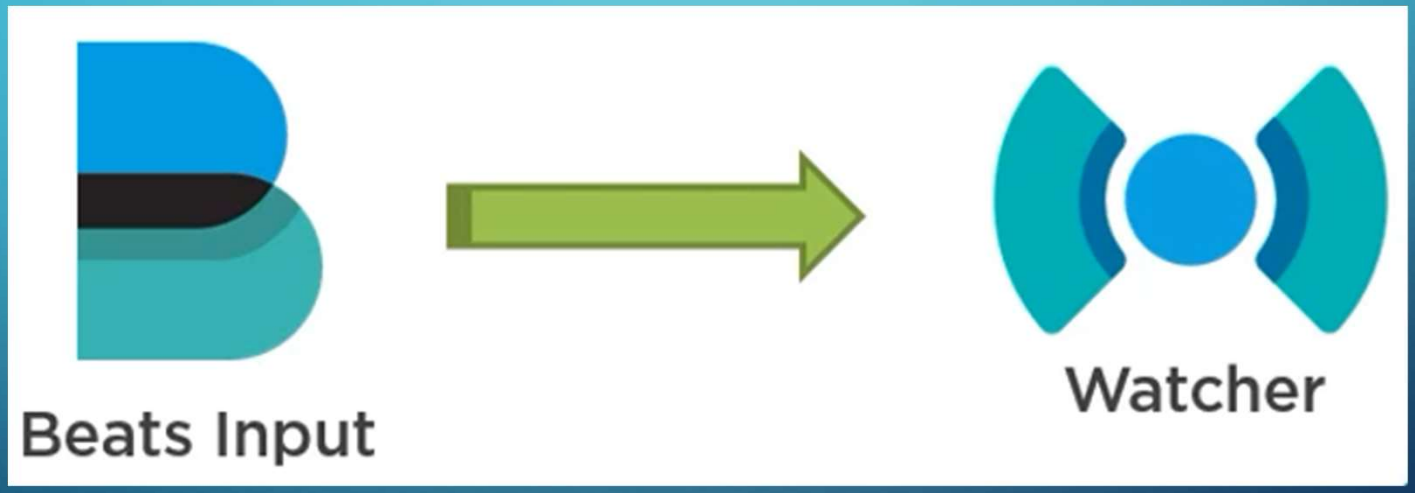
BLUE TEAM DEFENSE

*"... **I ALWAYS FEEL LIKE SOMEBODY'S WATCHING ME***
AND I HAVE NO PRIVACY "

-MICHAEL JACKSON

Created by Skye for UPenn Cybersecurity Bootcamp May 2022

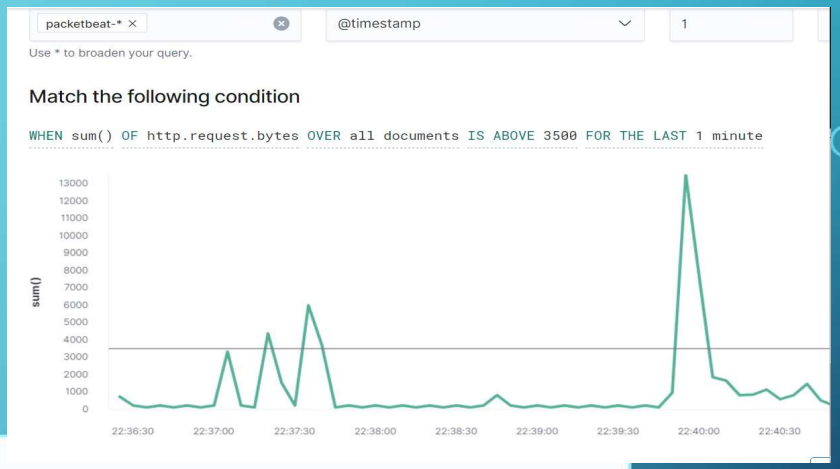
ALERTS IMPLEMENTED ON KIBANA WATCHER

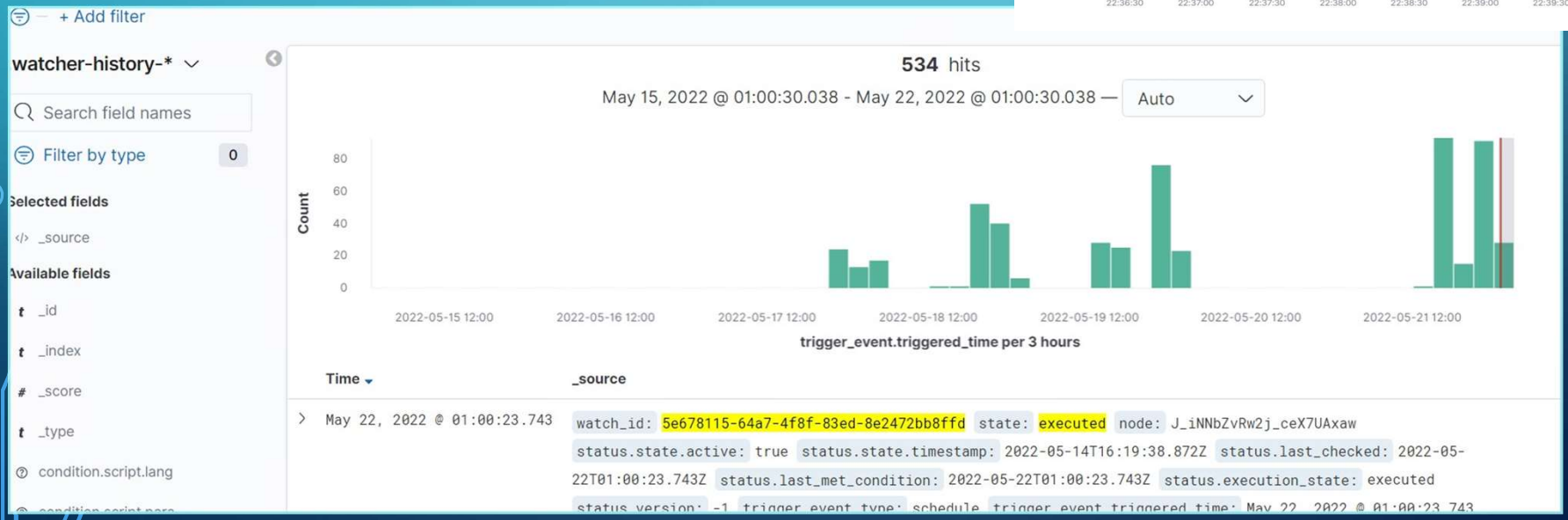


ALERT 1: HTTP Request Size Monitor

THE METRIC: http.request.bytes

THE THRESHOLD: when the sum of the bytes over all documents exceeds 3500 for the last 1 minute

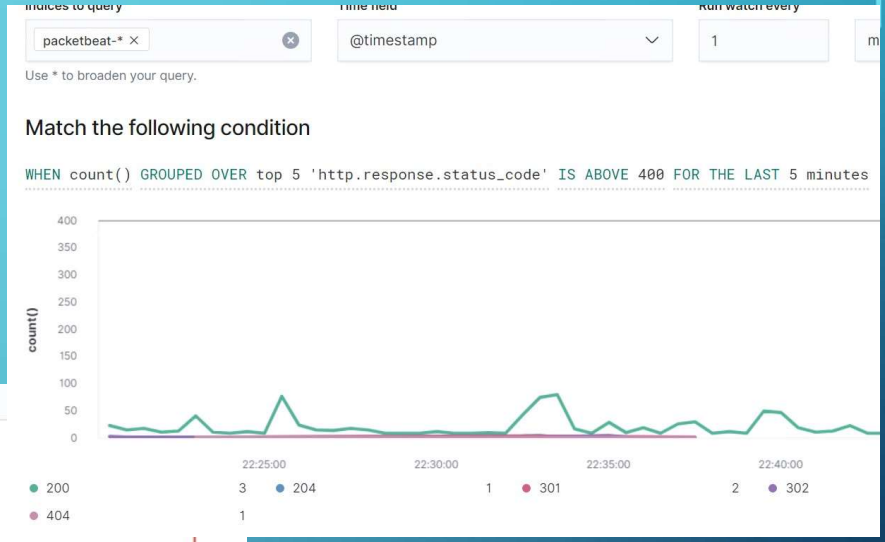




ALERT 2: Excessive HTTP Errors

THE METRIC: http.response.status_code

THE THRESHOLD: when the count of HTTP responses over 400 is grouped over top 5 for the last 5 minutes.

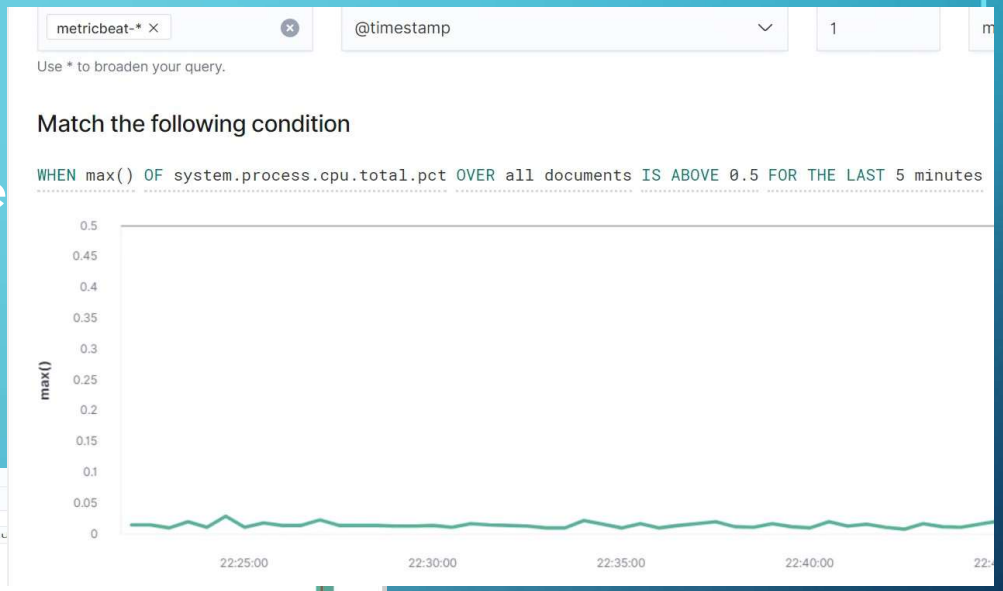


ALERT 3: Excessive CPU Usage

THE METRIC:

system.process.cpu.total.pct

THE THRESHOLD: when the percentage of CPU spent by the processes over all documents exceeds 0.5 for the last 5 minutes.



OTHER BEATS AND MODULES TO EXPLORE FOR MORE ALERTS

USING A MYSQL MODULE THROUGH METRICBEAT TO SPECIFICALLY IDENTIFY MYSQL TRAFFIC:

HTTPS://WWW.ELASTIC.CO/GUIDE/EN/BEATS/METRICBEAT/7.0/METRICBEAT-MODULE-MYSQL.HTML

USING INDICES LIKE AUDITBEAT AND LOGS-ENDPOINT.EVENTS TO SPECIFICALLY IDENTIFY NCAT TRAFFIC:

HTTPS://WWW.ELASTIC.CO/GUIDE/EN/SECURITY/CURRENT/NETCAT-NETWORK-ACTIVITY.HTML

HARDENING

Hardening Against WPScan on Target 1

Modify the NGINX Configurations/ Directives to Block WPScan:

- Block access to the xmlrpc.php file in the root folder:
  - **Why it works**: It prevents Remote Procedure Call

```
location ^~ /xmlrpc.php {
    deny all;
    error_page 403 =404 / ;
}
```

- Block WordPress Plugin Enumeration:
  - **Why it works**: It prevents WPScan from knowing the plugin versions to then scan the readme files inside those plugin subfolders.

```
location ~* ^/wp-content/plugins/.+\.(txt|log|md)$ {
    deny all;
    error_page 403 =404 / ;
}
```

- Block all Access to Install.php and Upgrade.php:
  - **Why it works**: It prevents WPScan from enumerating version query strings like Javascript Files.

```
location ^~ /wp-admin/install.php {
    deny all;
    error_page 403 =404 / ;
}

location ^~ /wp-admin/upgrade.php {
    deny all;
    error_page 403 =404 / ;
}
```

Hardening Against Unauthorized Access to MySQL on Target 1

Secure the wp-config-file by:

- Using newly generated Secret Keys
- Moving the wp-config.php file to somewhere other than the root folder of the site
- Block access to the wp-config.php file by creating an htaccess file inside the same directory

  How to: First create the .htaccess file (You might have to start with a htaccess.txt file then you would change it to a .htaccess file after editing.) and then edit the file to include:

```
1 <files wp-config.php>
2 order allow,den
3 deny from all
4 </files>
```

Why it works:

These techniques all harden the accessibility of the wp-config file. If an attacker cannot access the wp-config.php, they would not be able to see sensitive data saved there. In our case, the red team was able to view the username: *root* and the password: R@*v3nSecurity and this is what gave us access to MySQL!*

Hardening Against Weak Passwords On Target 1

- **Enact a Strong Password Policy:**
  - Require Complexity– length and varied types of characters
  - Require Changes/Updates to Passwords on a set schedule
- **Use a custom implementation of password hashing inside the wp_users file:**
  - Change the default (MD5 or SHA1) hashes with a more complex hash style using an “off-the-shelf” solution like password_hash():

```
password_hash(string $password, string|int|null $algo,
array $options = []): string
```

- **Why it works:**
  - According to CompTIA, one of the easiest ways for a password to be compromised is simply by guessing it. That is exactly what the Red Team did when they guessed the credentials for the user: *michael* to be the password: *michael*. A strong password policy for users would mitigate that risk.
  - When the user credentials were found in the wp_users file during the MySQL session, they were in a form that was easily broken in minutes by the service John the Ripper. Stronger password hashes would prevent that ease of access.

Hardening Against Brute Force Attacks On Target 1 And 2

- **Install a WordPress Firewall Plugin at the DNS or Application Level**

  - **Why it works**: A firewall service will filter and block nefarious traffic from accessing the site.

- **Make Sure WordPress is up to date:**

Run	`wp core version`	to verify current WP Version
Then Run	`wp core update`	to update to the latest version

  - **Why it works**: Having an updated WordPress Installation should provide security from older known vulnerabilities that have been patched in the most recent version of WordPress.

Hardening Against .php Uploads Through Local File Inclusion On Target 2

- Require Authentication to upload files.
  Examine the code used to upload files to make sure that the ‘move_uploaded_files()’ function will not be executed unless it is being done by an authenticated user.

- Block access to PHP files in a .htaccess file configuration:
  Inside the folder where access needs to be blocked:

```
<Files ^(*.php|*.phps)>
   order deny,allow

   deny from all

</Files>
```

Why it works:

‘move_uploaded_files()’ is the function that usually handles uploads like a PHP Shell. By finding, examining, and strengthening each application code that calls for ‘move_uploaded_files()’ upload access can be mitigated.

Hardening Against GoBuster Web Server Enumeration On Target 2

Using mod_rewrite on Apache to rewrite anything that looks like a request for a directory into a request for a file that doesn't exist:

```
$ cat htdocs/.htaccess
RewriteEngine on
RewriteRule ^([a-zA-Z0-9_/-]+)/?$ $1  [L]
```

Why it works: Since directory-enumeration attacks like GoBuster rely on searching a site from a file of popular director names, it can be assumed that any directory that does not return a 400 HTTP Response code exists.

By configuring Apache to rewrite anything that looks like a request for a directory to instead look like a request for a file that doesn't exist, then the response should be a 400+ error which would make the attacker think that the file does not exist.

IMPLEMENTING PATCHES

WE ARE GOING TO FIX THIS LEAKING WEB SERVER!

IMPLEMENTING PATCHES WITH ANSIBLE

- Ansible containers and automation services like Cron can be vital in the deployment, updating, and hardening of a Word Press web server.
- Publicly available Ansible Playbooks:
  - For WordPress:
    - https://github.com/do-community/ansible-playbooks/tree/master/wordpress-lamp_ubuntu1804
  - For Apache2
    - https://github.com/do-community/ansible-playbooks/tree/master/apache_ubuntu1804
- Keeping Apache and WordPress Up to Date, in general:

For Apache:

```
$ apache2 -v
$ sudo add-apt repository
ppa:ondrej/apache2
$ sudo apt update
$sudo apt install apache2
$ sudo systemctl restart apache2
```

For WordPress:

```
$ wp core version
$ wp core update
```