Simple Web Server

Michael Eaton
Victoria Zheng

Git Repository: git@github.com:zhengc/SimpleWebServer.git

Tactics for Security:

1. Limit Access

A blacklist has been added to the server in order to facilitate access limitations for hosts that have been determined unsafe. This list can be manually modified by the maintainer of the server, and has methods that allow a maintainer or developer to dynamically add hosts that have been determined to be engaging in unsavory conduct to the list.

2. React to Attacks

The Simple Web Server has been made to detect attacks by determining if a specific host has made 10,000 requests in the last minute. If this is the case, the server is added to the blacklist, causing all requests to automatically kick back a 403 forbidden error. This should, in effect, nullify a specific host from continuing to attack the server.

3. Inform Actors

The Simple Web Server has had logging of operations added to it which should allow maintainers to determine if a specific action has been taken by the server, or if any threats have been detected. By using log4j, it has the potential ability to email certain errors to interested actors so they can take proper actions.

Tactics for Performance:

1. Processing Time

Requests that do not generate codes that may result in malicious or long-standing requests have been allowed to keep their TCP sockets open following successful serving of the request. Codes that allow this to happen include 304, 404, and 200.

2. Bound Execution Times

If a request seems to be taking too long to process, the server will return a 504 Gateway Timeout. The time limit has been set at 10 seconds. This should increase performance dramatically.

3. Available Resources

The availability increases presented following performance have increased the number of resources available to the Simple Web Server, increasing performance.

4. Control Resource Demand

Large files (over 1GB) are automatically denied to ensure that hard-disk, memory, and network resources are not over-utilized in serving oversized requests. This should drastically reduce network traffic.

Tactics for Availability:

1. Monitor Fault Detection

The Simple Web Server has had logging added to it, which will alert any maintainers of faults that have occurred in the system.

2. Sanity Check

The Simple Web Server has had checks for files that are simply too large added, as well as checks to determine the frequency with which a host requests files. If either of these values are too large, the request is terminated with a 403 forbidden code. This should increase the availability of the server dramatically.

3. Exception Handling

Many common exceptions for the server have been handled in code.

Introduction to testing data and methodology:

Testing was performed using the SimpleWeb project in its vanilla form (Pre-Modification), the modified SimpleWeb the team produced (Post-Modification) and the Basic DOS attack client provided with the project over a wireless-N connection provided from a standard NetGear router. This likely is the culprit of much of the data's variations, and the large amount of Jitter in the Post-Modification data should show this.

The data collected is contained in .csv files in the git repository. This data is a series of millisecond values denoting the average turnaround time for GET request for a 15-Byte file during a 15-minute test of the SimpleWeb systems using the Basic DOS client. The data was then imported into Microsoft Excel 2010 and data was mined from it in that fashion. This, rather unfortunately, resulted in truncation in the datasets from the limitations of Excel, however the data provided should even out over a larger dataset.

Availability Data:

1. Pre-Modification Availability

MTBF: 15 minutes

(No failures encountered between the 15 mins that the server was running)

MTTR: 2.37 ms

Availability: 99.99%

2. Post-Modification Availability

Note: These tests were performed with attack detection and blacklisting off.

Much of the degradation in performance can be attributed to the additional logging that was added.

MTBF: 15 minutes

(No failures encountered between the 15 mins that the server was running)

MTTR: 2.45 ms

Availability: 99.99%

Performance Data:

1. Pre-Modification Performance

Latency: 1.24 ms

Jitter: 2.91 ms

Throughput: (15B/1.24ms) = 12.1KB/s

2. Post-Modification Performace

Latency: 1.67 ms

Jitter: 8.88 ms

Throughput: (15B/1.67ms) = 8.98KB/s

Security Data:

1. Pre-Modification Data

Detection: Infinite. The original project contained no code to detect security breaches

Reaction: Infinite. The original project contained no code to react to security breaches

Recovery: Infinite. The original project contained no code to recover from security breaches. A case can be made supporting the MTTR as the recovery time.

2. Post-Modification Data

Detection: 16.7s

Dependent purely upon the time between packet sends from the attacker. The server will detect when more than 10000 requests have been sent from a single host in under a minute. It can be reasonably extrapolated from the Latency Data that a time of (1.67 ms * 10000) = 16.7 seconds will be required before a threat is detected.

Reaction: ~1.67ms.

Once the threat is detected the offending IP is added to a blacklist. This IP will then be denied requests for the rest of the server's uptime. This process should take roughly the same amount of time as a standard response.

Recovery: Instantaneous

Once the reaction has been completed, the server is fully recovered.

Notes on Functionality:

The extensive logging involved in the project likely slowed it more than truly necessary. However, this was necessary to provide the user with enough output to reasonably determine performance of the server. And the amount slowed down by adding logging was not significant enough to outweigh the convenience that was added by having detailed logging so we kept it that way.