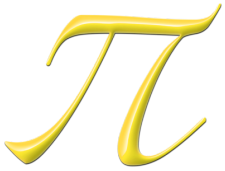

Paragon Initiative Enterprises

ByteJail Backend

Source Code Audit
20-21 June 2015

I. Introduction

This document describes the security audit of the TahoeSmart and TahoeWorker projects of Bytejail by Paragon Initiative Enterprises for Christian Hermann (bitBeans).

Our audit targeted git commit `22e71cb720cb96f754a8e78447bf0d756a7cd050`, which was committed on June 18, 2015.

This report was prepared by Scott Arciszewski, CDO, and reviewed by Robyn Terjesen, CEO.

Audit Results Summary

After a comprehensive code review of the TahoeSmart and TahoeWorker projects, we did not identify any security vulnerabilities in either project. However, our investigation did uncover a few helper classes that could benefit from security enhancements.

II. Audit Scope

We have limited the scope of this audit to focus specifically on the two Bytejail projects that were provided:

- TahoeSmart, the bytejail API based on Nancy and Tahoe-LAFS gateway
- TahoeWorker, the bytejail backend worker

We excluded the following from our scope (although we did verify that their features were being used safely):

- Microsoft .NET Framework
- Nancy Web Framework
- Devart's MySQL connection driver
- The Sodium cryptography library (and its .NET bindings)

III. Issues

No security vulnerabilities were discovered in the TahoeSmart or TahoeWorker projects.

IV. Other Findings

Note: The findings in this section are not necessarily vulnerabilities.

1. Consistently use CSPRNG for new Jail names

The `GetJailName` and `GetRandomString` methods in the `TahoeSmart.Helper.Utils` class depend on another method called `GetRandomNumber`. Both methods were marked for migration to libsodium-net (pending a stable release with `Sodium.SodiumCore.GetRandomNumber`).

`GetRandomNumber` obtains 4 bytes from a cryptographically secure pseudorandom number generator, converts them into an integer with bit-shifting and binary OR operations, then seeds the (non-cryptographically secure) `System.Random` class with the 32-bit integer derived from the CSPRNG, and uses that for the output.

Additionally, when `GetJailName` invoked `GetRandomNumber`, it did so to select a random value between 1 and 30 out of an array with indices ranging from 0 to 30. As a result, the 0th value in the array would never appear in a randomly generated jail name.

We provided a stop-gap solution that generates random integers from the system CSPRNG until the next stable release for libsodium-net is available. Our solution was accepted before this report was completed.

2. Escape Variables Before Concatenating with Shell Command Arguments

There are many points in both applications where the Shell.ExecuteShellCommand helper is invoked. Although this helper separates the arguments from the command, there were several places which a variable was concatenated with the argument string. None of these variables were escaped for shell meta characters. (Fortunately, none of the variables we analyzed can be controlled by an attacker on the network.) For example:

```
Shell.ExecuteShellCommand("/bin/sh", "/root/secure/tahoe-
worker/sign_bjcfg.sh " + connectionConfigPath);
```

Because passing arguments to `/bin/sh` allows multiple commands to execute, if an attacker could remotely set the connectionConfigPath to something like "`/some/file; rm -rf / &`" (without the previous file existence check failing, which is unlikely), they could make the TahoeWorker process attempt to delete the entire filesystem.

We provided a patch that adds and utilizes an `Escape()` method to the Shell helper in both projects. Our patch was merged before this report was completed.

V. Conclusion

After carefully reviewing the source code for both projects, we are highly confident that the Bytejail backend projects, TahoeSmart and TahoeWorker, are secure against both remote and local attackers.

Every Nancy API endpoint we reviewed was consistently validated by libsodium's public key signature verification, which means even getting a forged API request to be accepted by the application, an attacker would need to either break the Ed25519 asymmetric signature algorithm or steal a private key from a legitimate client. Even when armed with a stolen key, we did not identify any ways for an attacker to forge a request to take over the server running either project.

Of our two findings, one was already known by the developer and the other was not exploitable due to application logic. If, going forward, the Bytejail development team meticulously escapes all variables before concatenating with the arguments string, remote code execution vulnerabilities will never surface in the Shell helper. Given the extremely high quality of both projects, we don't anticipate they will overlook any such instances in their final product.