I. Introduction

This document describes the third party software security assessment of the Qbix Platform, conducted by Paragon Initiative Enterprises.

Our investigation targeted Git commit `3091c67e6b1ef945b09689486e86cfb65f51d1b6` of the Qbix Platform. We did not examine the security of any additional Qbix modules.

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

Report Summary

This audit took place beginning on March 4, 2018 and concluded on April 2, 2018. In that time, only 4 security vulnerabilities were discovered in the Qbix Platform. This result is a testament to the developers' dedication to application security best practices.

- 2 with a severity rating of **medium**
- 2 with a severity rating of **low**

II. Vulnerabilities in the Qbix Platform

1. Insecure Randomness Used for Nonce

Severity: **Medium**

Qbix includes a nonce feature for validating HTTP forms, in accordance with the standard CSRF attack mitigation strategy.

However, we were able to discover a weakness that would allow this nonce to be predicted by an attacker, thereby defeating the CSRF protections and allowing third-party web pages to execute commands on behalf of an authenticated user.

In platform/classes/Q/Session.php on Line 927, the nonce is generated as follows:

```
$_SESSION['Q']['nonce'] = sha1(mt_rand().microtime());
```

This combines an insecure random number generator called a Mersenne Twister with the current time to microsecond precision, then hashes these two values with the broken hash function SHA1.

The number of possible outputs for this function is approximately 2^{53} (assuming the time of day is not known), which is within the capabilities of a modestly determined attacker. This is the optimistic attack complexity; in reality, an attacker who studies 624 sequential outputs of the Mersenne Twister algorithm used in PHP can predict every subsequent value until it's reseeded.

PHP 7 introduced a function called `random_bytes()` which transparently uses the best possible source for randomness available on the current operating system and hardware. Paragon Initiative Enterprises developed a PHP 5 polyfill for the `random_bytes()` and `random_int()` functions is available on Github, called random_compat.

Recommendation

We included a patch for this and related vulnerabilities for the Qbix Platform, which integrates with the standard PHP package manager, Composer, to load in the latest versions of random_compat and other security libraries.

Resolution

A modified patch (one that does not adopt Composer) was applied by Qbix, Inc. Our team confirmed that this vulnerability is no longer present in Qbix.

2. Timing Side-Channel in Session Cookie Signature Validation

Severity: **Low**

If an external secret was configured, the Q_Session class uses HMAC-SHA1 to ensure the session ID was not tampered with by a malicious end user.

In platform/classes/Q/Session.php lines 1102-1104, this authentication code is validated like so:

```
$c = isset($secret)
    ? ($b === substr(Q_Utils::signature($a, $secret), 0, 32))
    : true;
```

The use of strict equality (=== instead of ==) does successfully side-step type-confusion issues which are common in older PHP cryptography libraries. However, it is still vulnerable to a timing analysis side-channel that could allow skilled attackers to bypass this security feature.

Recommendation

Included in our patch for the first issue was a `hash_equals()` polyfill for versions of PHP older than 5.6, and the Q_Session class was updated to mitigate these vulnerabilities.

Resolution

A modified patch (one that does not adopt Composer) was applied by Qbix, Inc. Our team confirmed that this vulnerability is no longer present in Qbix.

3. Insufficient Password Hashing Security

Severity: **Low**

Qbix's password hashing function, as defined in platform/plugin/Users/classes/Users.php on lines 1402 through 1445, uses something similar to PBKDF2 -SHA1, except without HMAC, and a default iteration count of 1103. This function also uses an insecure random number generator for generating salts.

However, the security profile of this function is probably closer to PBKDF1 than PBKDF2, since it's only using one character of a **confounder** (which is simply the password and salt, concatenated) with the previous iteration's output for each round. Furthermore, the way this function is written only allows two internal hash functions to be selected: MD5 and SHA1. Neither of those two should be relied on for security.

Recommendation

Given the use of weak randomness for salt generation, the use non-standard password hashing algorithm, the weak underlying cryptographic hash function, and the low default number of rounds, we recommend migrating towards an industry standard password hashing function, such as:

- Argon2id, with 64 MiB of memory and 2 rounds
- bcrypt, with a cost factor of at least 10
- PBKDF2-SHA512 with at least 100,000 rounds

Since our previous patch provided Composer, we offer an additional patch that migrates password hashes to use the password hashing API introduced in PHP 5.5, using a polyfill library by the author of that PHP feature for PHP 5.3.7+ websites.

This offers bcrypt with a default cost of 10, although this cost factor can be tuned to achieve better performance or better security.

Resolution

A modified patch (one that does not adopt Composer) was applied by Qbix, Inc. Our team confirmed that this vulnerability is no longer present in Qbix.

4. Usernames Not Validated or Sanitized When Generating URLs

Severity: **Low**

When a user first registers with Qbix, if they opted to supply their own username, it is used without sanitation or validation when generating their URL.

```
$user->url = $username ? "http://$username.".$url_parts['host'] : "";
```

This value is in turn used for generating redirect URLs, e.g. in the User_authorize_response() function. We did not find evidence of this being a risk factor for cross-site scripting (XSS) attacks.

Nonetheless, we highly recommend validating that `$username` is a valid subdomain before concatenating it with the app's hostname. We've included a patch with a recommended fix.

Resolution

Our team confirmed that this vulnerability is no longer present in Qbix.

IV. Miscellaneous Remarks

Important: The items in this section are not security vulnerabilities.

1. Package Management

In one of our patches, we introduce a methodology for Qbix to adopt the standard PHP package manager, Composer. Most of the exciting work in the open source PHP ecosystem is facilitated by Composer.

By using Composer, Qbix can quickly update, test, and release upstream security updates to the platform should any be discovered in any of the open source libraries that it depends on. All the team needs to do is configure the version constraints in the composer.json file appropriately, then run one command:

```
php composer.phar update
```

No more manual updating and patching. Adopting Composer will be a boon for productivity and security. We highly recommend it.

Resolution

This issue is unresolved due to the business model and target audience of Qbix, Inc. depending in part on PHP 5.2 support, whereas Composer requires a minimum of PHP 5.3.

Seeing as the requirements for resolving this security concern are at odds with the business needs of the target audience of Qbix (i.e. PHP 5.2 users who for whatever reason cannot or will not upgrade to 5.3+), there is little left for us to recommend from a security standpoint.

The only remaining avenue is to plead to these users to upgrade to a secure version of PHP and hope enough of them do so that Qbix can adjust their priorities.

2. Certificate Authority Bundle Management

In several places, HTTP requests are made using cURL but certificate validation is configured incorrectly.

This is not listed as a security vulnerability, because this was already known to the development team, as demonstrated by the comments on RedactedClass`::createCurl()`:

```
//========= NEED TO REMOVE THESE LINES LATER! THIS IS SECURITY HOLE
==============//
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, FALSE);
curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, FALSE);
//========= ==================================================
==============//
```

The values you want for each configuration option is as follows:

CURL constant	Recommended Value
CURLOPT_SSL_VERIFYPEER	TRUE
CURLOPT_SSL_VERIFYHOST	2

There are two things you can do to make it easier to keep everything secure and minimize troubleshooting.

First, switch from cURL to Guzzle. It's a far easier-to-use HTTP library than working with cURL directly.

Second, consider adopting Certainty, a project we created with the aim of ensuring everyone has the most recent copy of the Mozilla certificate authority bundle, even on esoteric operating systems that put the standard cacert.pem file in a non-standard location.

Resolution

Qbix has not yet made steps to migrate to a higher-level HTTP library like Guzzle or adopted a library like Certainty to ensure they can always use the most-secure HTTPS client configuration for server-side requests.

However, this isn't an issue for the Qbix core (which doesn't send data to third-party servers), only specific plugins and/or apps built with Qbix.

V. Conclusion

Our security team scrutinized the Qbix source code heavily, looking for weak points that could lead to a loss of confidentiality, integrity, or availability.

Despite a full month of security analysis from our company, we only turned up a mere four security vulnerabilities that were not already known to their developers, which were promptly fixed by the Qbix developers.

We identified two additional points of concern that are not trivially solved: Package management and server-side request security. If package management can be solved, then server-side request security is easy to address in turn.

We are confident that Qbix's security is more than adequate to protect their users from malicious or accidental data leakage.