J Comput Virol (2008) 4:347–356
DOI 10.1007/s11416-007-0075-8

INVITED PAPER

SQL infections through RFID

Anthonius Sulaiman · Srinivas Mukkamala · Andrew Sung

Received: 5 October 2007 / Accepted: 23 November 2007 / Published online: 18 December 2007

Abstract Automatic identification and collection (AIDC) technologies have made the life of a man much easier on numerous platforms. Of the various such technologies the radio frequency identification devices (RFID) have become pervasive essentially because they can track from a greater physical distance than the rest. The back end that supports these RFID systems has always been working well until they encounter a sbadly-formatted RFID tag. There have hardly been any incidents where such tags, once identified by the back-end systems, can in fact wreak havoc via the interacting databases in the RFID infrastructure. Recently, there has been significant research in this area. In the previous work, the author managed to do an attack using a self-referential query on Linux, Oracle, and PHP. However, they have been unable to test it on SQL Server 2005. This paper differs from the previous work in the way that it extends the attack using a self-referential query to Windows, SQL Server 2005, and ASP with their respective latest updates installed. The query itself is more robust by making certain that the table can contain it.

Keywords RFID attacks · RFID virus · SQL infections · SQL injections

A. Sulaiman (✉) · S. Mukkamala · A. Sung
Department of Computer Science,
Institute for Complex Additive Systems Analysis,
New Mexico Tech, Socorro, NM, USA
e-mail: ais@cs.nmt.edu

S. Mukkamala
e-mail: srinivas@cs.nmt.edu

A. Sung
e-mail: sung@cs.nmt.edu

1 Introduction

Essentially, it is the ease of tracking of various objects simultaneously from a distance that made the RFID technology become so prevalent. Also, the fact that access control can be enforced caused numerous business establishments to implement this technology in one way or the other, either for supply chain management or to keep a track of the employees by embedding an RFID tag in their ID cards [1].

Similarly hospitals also keep a track of their doctors and also trying to read medical data from sub-dermal RFID in the patients RFid Gazette [2]. Numerous highway maintenance authorities use RFID tags inserted on the vehicles for easy and automated toll collection New Jersey [3]. More recently efforts are underway by government authorities to introduce these RFID chips citizens' passports [4].

RFID systems have been around since the World War II used by many nations' military to keep a track of their artillery. As stated earlier, the RFID systems are a part of the AIDC technologies which also consists of other systems like the optical character recognition (e.g. barcodes) and magnetic ink character recognition (e.g. magnetic stripe). An estimated of 40 million people in the US are supposedly carrying some kind of RFID device with them [5]. This relatively shows the ubiquity of the RFID devices. These devices are controlled by corresponding computer infrastructure. In most cases a database exists to contain the individual details of the tags.

This technology has its down side too. There are various security and privacy issues. Though privacy issues are not of concern in this work, Sect. 2.1 illustrates a possible real world scenario that can compromise privacy due to lack of proper security of the technology. This work focuses on an RFID tag triggered executable on a Windows machine using Microsoft SQL Server 2005. While there are works focusing

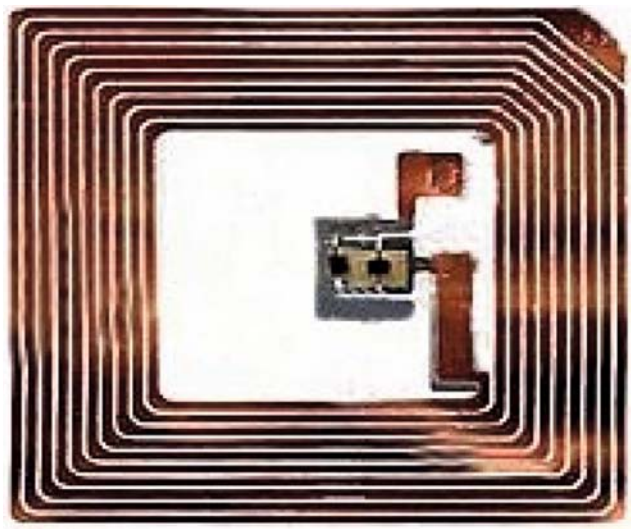


Fig. 1 A RFID tag

on Linux using Oracle (both are freely downloadable), we focus on commercial products and how they perform when these types of attacks occur. This paper presents an RFID attack on Microsoft SQL Server 2005 that had not been successfully performed until now. We also extend the work by making a more robust SQL query.

The following section explains the basic RFID technology followed by possible threats. Section 3 explains the related works and Sect. 4 talks about the RFID virus that this work is concentrated on. Section 5 presents the experiments and results of this work and finally in Sect. 6 concludes the paper with possible future work.

2 RFID technology

RFID systems primarily consists of the RFID tags or chips, the RFID readers, the antennas, the computer networks and finally the software that takes care of the information carried by these RFID tags. An RFID tag (shown in Fig. 1) is a tiny, flat microchip with a built-in antenna, which is available in various sizes, but with the same basic functionality.

When a radio signal is incident on an RFID tag, the RFID is activated and broadcasts the information it contains. The RFID tags can be attached to or incorporated into a product, animal, or person for the purpose of identification using radio waves [6].

These RFID tags are of two types: passive and active. The passive ones do not have any power of their own. They respond only if encountered by radio waves from the readers. On the other hand, active RFID tags have a built-in power source and more recently, certain computing or sensing technology (e.g. sub-dermal chips) which emit the data in the form of radio frequency waves to be received by a legitimate reader. These active RFID tags usually have higher storage capacities when compared to their passive counterparts. They usually provide communication ranges of 100 m or more while passive can only provide up to 3 m [7].

Table 1 shows the differences of passive and active tags taken from [7].

We are concerned with the back-end entity of the RFID infrastructure which is the software aspect essentially the databases. The databases have their own set of security issues. With the rise of internet usage and computing power, a lot of information from these databases is exchanged over the internet. Hence, these databases are at the verge of facing the various security problems that engulfed the internet.

2.1 Threats

RFID systems encountered a number of threats and privacy issues. This work is an attempt to address the information security issues chiefly the attacks through the databases that these RFID tags communicate with. This paper provides a proof-of-concept to run an executable on a windows machine using SQL Server 2005 as the back-end architecture.

The authors of [8,9] enumerate a number of common threats that can implement the above mentioned method. As an example, the following classes of problems especially are of major concern for both security and privacy in the RFID tags incorporated in an individual's problems:

- *Identity spoofing*. An attacker replaces an authorized reader with their reader and reads the tags of an individual without the individual's authorization [8].
- *Data tampering*. Products often have RFID-enabled tags to store pricing and item information. An attacker modifies the tag to show cheaper price when scanned at the cashier.
- *Repudiation*. The government may use RFID-enabled tags to track, hot-list, or profile individuals [8]. The government has already controlled people with RFID-laced bracelets—and not just criminals [10].
- *Information disclosure*. An attacker tracks an individual determining where an individual is located and where they have been by the tags carried by an individual being read at multiple locations [8].
- *Denial of service*. An attacker deletes or modifies the serial number in an RFID-enabled passport preventing or delaying the individual from entering the country [8].
- *Elevation of privilege*. An attacker modifies the serial number on a RFID enabled passport to be a citizen in good standing instead of a criminal [8].

2.2 RFID risks

Though RFID technology has brought lots of usages and significantly changed the business process, it has its own risks.

Table 1 Differences of active and passive tags

	Active RFID	Passive RFID
Tag power source	Internal to tag	Energy transferred from the reader via RF
Tag battery	Yes	No
Availability of tag power	Continuous	Only within field of reader
Required signal strength from reader to tag	Low	High (must power the tag)
Available signal strength from tag to reader	High	Low
Communication range	Long range (100 m or more)	Short or very short range (3 m or less)
Multi-Tag collection	Collects thousands of tags over a 7 acre region from a single reader	Collects hundreds of tags within 3 m from a single reader
Sensor capability	Ability to continuously monitor and record sensor input; date/time stamp for sensor events	Ability to read and transfer sensor values only when tag is powered by reader; no date/time stamp
Data storage	Large read/write data storage (128 KB) with sophisticated data search and access capabilities available	Small read/write data storage (e.g. 128 bytes)

For example, suppose an attacker tries to compromise the system or damages the database using SQL injection attacks, there would be a heavy loss in business. These risks can be classified as high-level business risk [11]:

- *Business process risk*. The usage of RFID systems has made lot of things easier. For example, RFID-enabled systems replace the paper-based inventory management system in warehouses [11]. It is because the paper system might be more resilient to local disaster than the RFID system. When there are any failures in RFID system such as network failure, middleware infection by virus or any interruption of signals while data is transmitting, the data is less accurate. This, in turn, might sometimes lead to even more critical stages of devastation to the business. This type of risk is a cause from both human action and natural causes, which might be intentional or unintentional. An example of unintentional risks is when a tag fails to be read due to some damage that occurred to the RFID tag. An example of intentional risks is when an attacker clones the tag.
- *Business intelligence risk*. An attacker can gain an unauthorized access over the RFID system and get sensitive information such as password and other information from the back-end database system. For example, an attacker eavesdropping on the RF signals can capture the data. Supply chain applications are most vulnerable to this kind of attack [11].
- *Privacy risk*. In the case of when a customer purchases a merchandise and the tag is not removed or destroyed, the tag is still in active state. The seller might still use the tag which stays on the customer. He can get the customer's location by tracking the tag's location. He can even get some personal information about the customer with the help of the tag [11]. Privacy also depends on how the information is stored in the tag, the built-in security of the tag (e.g. encryption) and how secure the database systems and the middleware.
- *External risk*. External risks always exist, since the RFID systems are sometimes connected to non-RFID systems or external network to connect to an enterprise system or database servers. The major problems involving network devices and applications are network attacks [11]. These problems are caused by malware and vulnerabilities present in the network devices and the middleware systems (applications). These vulnerabilities lead to damaging the database system and compromising the whole enterprise system.

2.3 Attacks on RFID

The following defines data security in an RFID context [12]:

- *Controlled access to the data:* Authentication needs for read and write access over information.
- *Control over access to the system:* All devices connected to the system are authentic and trustworthy.
- *Confidence and trust in the system:* There is a general perception that the system is safe and secure.

RFID technology has its own disadvantages in security and privacy aspects. The security of an RFID-enabled system depends mostly on how secure the middleware is developed. It also relies on the data contained in RFID tags, which can surprisingly lead to a SQL injection attack, denial of service attack or even a buffer overflow.

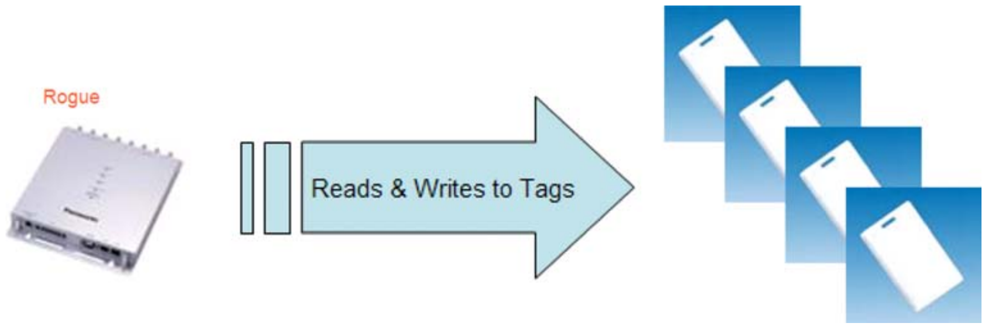


Fig. 2 Unauthorized tag access

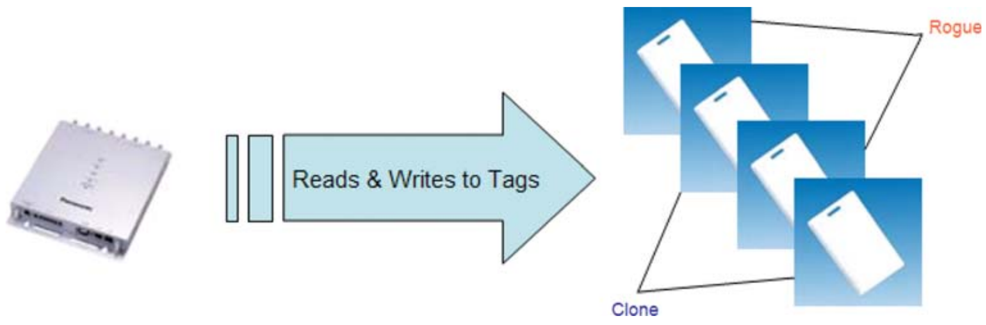


Fig. 3 Clone tags

Tag readers can communicate in two ways. There are several security issues based on these communication methods: when tag readers convey data via internet protocol and when tag readers provide and gather data to and from the tags via low power radio frequency.

Tag readers convey data via internet protocol (IP) An unauthorized access through the network is the key threat. No access should be allowed for any rogue devices to connect to the network. The network can be secured by using techniques like Secure Socket Layer (SSL) and Secure Shell (SSH). These techniques ensure more security by closing all open ports that can be used by intruders to gain access via telnet. Because of the availability of secure tools and standard feature techniques, the back-end communication is strong and somewhat secure. Hence, IP communication is an essential feature of an RFID reader and RFID implementations.

Tag readers provides and gathers data to and from the tags via low power radio frequency (RF) In this method, the communication is done on-air and can lead to several key threats:

- *Unauthorized tag access (leads to Sniffing)*. All tags are supposed to be read by an RFID reader which passes on by the authentication key. The rogue readers, an unauthorized reader, are similar to an RFID reader which can read any tags, as shown in Fig. 2. Because of a critical vulnerability in the tags, the rogue readers can read confidential information from the tags, write any malicious data into the tags and even can make inactive or kill the tags. Hence, tags can be read anywhere at any reachable distance. One recent controversy highlighting this issue concerns the skimming of digital passports (a.k.a. Machine Readable Travel Documents Biometrics [9,13].
- *Clone tags (lead to Spoofing)*. Clone tags are unauthorized replicas of the original tags, as shown in Fig. 3. These clones can be used to gain unauthorized access, since the readers will read these tags and assume to be the original ones. Rogue tags can be used to inject some false or malicious data into the system, which might damage the integrity of the system and the data in the system. One notable spoofing attack was performed recently by researchers from Johns Hopkins University and RSA Security [14]. The researchers cloned an RFID transponder using a sniffed (and decrypted) identifier that they used to buy gasoline and unlock an RFID-based car immobilization system [9].
- *Side channel attacks (lead to Replay attacks)*. The biggest threat is when there is a rogue device eavesdropping the communication between the tags and the reader, as shown in Fig. 4. The rogue devices can spoof the password and other sensitive information from the tags. Man-in-the-middle attack is also possible, where original data is modified and resent. For example, attackers can intercept and retransmit RFID queries using RFID relay devices [15]. Digital passport readers and contactless payment systems can also be fooled [9].

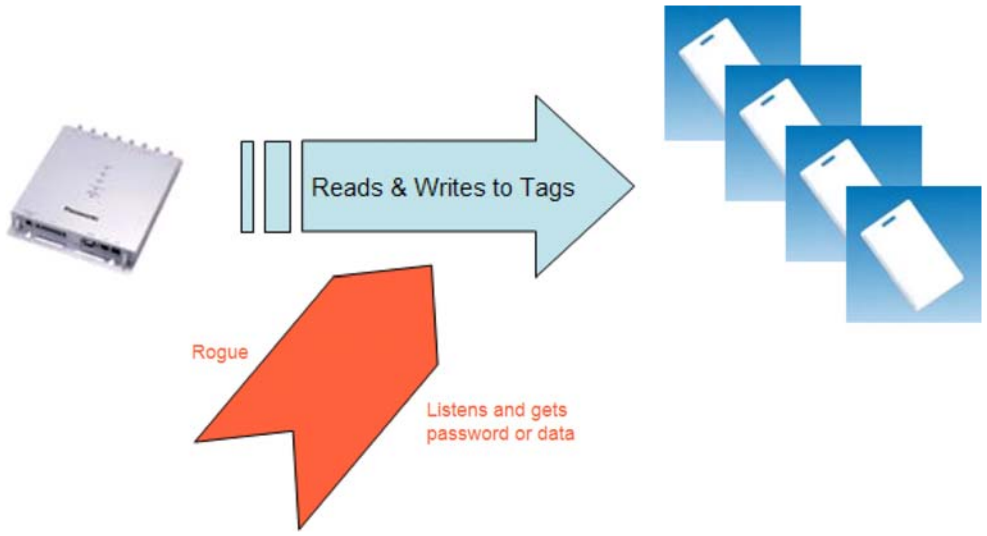


Fig. 4 Side channel attacks

2.4 SQL injection

Another threat related to RFID systems is SQL injection. SQL injection is a security vulnerability that occurs in the database layer of an application [11]. It is an attack technique used to exploit web sites by altering backend SQL statements through manipulating application input [17]. In the case of RFID systems, the vulnerability occurs mostly in the middleware. Middleware is the software that connects the RFID reader with the database system.

There are a few methods of SQL injection attacks, such as incorrectly filtered escape characters or incorrect type handling. Let us take a look at the most common mistakes that occur during the development of the middleware.

Our middleware reads from the RFID reader. Since this function does not relate to the database, there is no meaningful attack that can be done to the database here. The next step, however, authenticates the person trying to get in. The RFID tag contains the ID and the name of the person. The middleware authenticates with the following SQL query:

```
SELECT * FROM Table WHERE ID = $ID AND Name = ’$Name’
```

The middleware substitutes the $ID and $Name variables with the ID and the name read from the tag before submitting it to the database for execution. If the query returns a non-null value, he or she will be granted access to open the door.

A SQL injection attack using incorrectly filtered escape characters would be having the following malicious string in the Name field of the tag:

```
a’ OR ’a’ = ’a
```

The resulting SQL query during execution is:

```
SELECT * FROM Table WHERE ID = 0 AND Name = ’a’ OR ’a’ = ’a’
```

This query when executed will always return a value, which in turn grants the person access to the door.

Manipulating string in a SQL query is widely practiced and really dangerous as it can lead to executing another query. Consider the following string [18]:

```
a’; DROP TABLE Table;--
```

The resulting SQL statement has two queries which will end up deleting the whole table. Now no one can get in through the door. This is the real life denial of service. It is becoming even more dangerous when the database is served live on the network or the internet, especially when SQL operations are done through the use of web sites and URLs [19]. We chose SQL injection attack in our work because this attack is very common among databases.

3 Related work

There has been very little work concerned with RFID tags injecting viruses into the backend systems. Rieback et al. have done significant work in this area while providing a proof-of-concept for both Linux and Windows based systems [9]. In their paper they target Oracle with Server-Side Includes (SSI) performing SQL injection and script based attacks. They used PHP along with SSI to achieve the above.

Quines can be used to obfuscate the source code [11]. For increased stealth and generality this obfuscation can be utilized. Essentially a quine is a program that produces its source code as its output. The authors of [9] also mentioned about this mechanism in their work.

SQL injection attacks have been around for quite a while. In this work we show how attackers can use this in conjunction with RFID middleware to compromise the infrastructure.

SQL Injection happens when a developer accepts user input that is directly placed into a SQL Statement and does not properly filter out dangerous characters. This can allow an attacker to not only steal data from your database, but also modify and delete it [21].

Rieback et al. [9] managed to do an attack using a self-referential query on Linux, Oracle, and PHP. However, they have been unable to test it on SQL Server 2005. This paper differs from their work in the way that it extends the attack using a self-referential query to Windows, SQL Server 2005, and ASP with their respective latest updates installed. The query itself is more robust by making certain that the table can contain it.

4 RFID virus

The intelligence of an RFID system lies in the back-end systems and the middleware. In general the RFID tags by themselves are non-computing devices. On an encounter with a RFID reader they reply with a number or string as described in Sect. 5. This replied data can be stored and queried in a database.

The middleware application is the one that talks to the database while creating SQL statements and inserting the relevant information into the database. At this juncture an RFID tag can exploit the vulnerabilities in the middleware to send malicious payloads as input through the RFID middleware and hence perform a SQL injection attack [9]. This work explores such class of attacks that could be used to propagate RFID malware.

5 Experiments and results

The computer that we are using was configured as follows:

- Operating System: Microsoft Windows XP SP2.
- Database: Microsoft SQL Server 2005.
- Web Server: WAMP5 1.6.6 (Apache 2.0.59 with PHP 5.2.0) running on port 80 and Microsoft Internet Information Server included in Windows XP running on port 9919.

We use Windows and SQL Server as these are commercial products that get regularly updated as new vulnerabilities are discovered. The triggering mechanism is done through vulnerability in Server-Side Includes (SSI) which is available in any web server that supports it. We use SSI in conjunction with ASP and PHP. PHP can be freely downloaded and installed. ASP comes with Windows web server. This is to demonstrate that as long as SSI is enabled with execution capability, it can be exploited.

Table 2 An uninfected table

ID	Data	Mode
1	Toyota	Camry
2	Chevy	Blazer
3	Ford	Taurus

5.1 SQL virus using a self-referential query for SQL Server

The first experiment is to ensure that the virus mentioned by Rieback et al. can be applied to SQL Server 2005. In our experiment, we consider an inventory table called *Cars* that consists of car information under the *Data* column and model of the cars under the *Mode* column in our database. Our table is configured with the following columns:

- ID, int, primary key.
- Data, varchar(50).
- Mode, varchar(50).

We populate the table with the information shown in Table 2 to represent an uninfected table.

To update the contents of a database table, we run the following query:

```
UPDATE %Table% SET %ColumnToChange% = ’%NewData%’ WHERE %TagID%=’%id%’
```

We based our SQL virus from Rieback et al. that uses a self-referential query. The modified query to work with SQL Server is as follows:

```
ALTER TABLE Cars ALTER COLUMN Mode VARCHAR(MAX) UPDATE Cars SET
Data=’Toyota’,Mode=(SELECT SUBSTRING(text,71,185)
FROM sys.dm_exec_requests CROSS APPLY sys.dm_exec_sql_text(sql_handle)
WHERE CHARINDEX(’<!--#exec cmd="C:/windows/notepad.exe"-->’,text)>0)--’
WHERE ID=1
```

The payload here is the execution of Windows Notepad. However, it can be any very dangerous shell command like FORMAT. The length of the query in SQL Server is considerably longer than that of Oracle. The total length of the query is 267 bytes, which may not be feasible to put into some RFID tags.

The reasons for this length are as follows:

SQL Server has a longer name for system tables. The queries’ handles are stored in sys.dm_exec_requests and the query texts themselves are stored in sys.dm_exec_sql_text. In order for the query to refer to itself, we must cross apply both tables.

Function names in SQL Server are slightly longer than their Oracle counterparts. There is a possibility that the target column does not have enough space to hold the virus. In our case, we only set the Mode column to fifty characters. In order to be able to infect the table, a modification is required. We do so by altering the table and the column that we intend

Table 3 An infected table

ID	Data	Mode
1	Toyota	Toyota',Mode=(select substring(text,71,185) from sys.dm_exec_requests cross apply sys.dm_exec_sql_text(sql_handle) where CHARINDEX('<!–#exec cmd="C:/windows/notepad.exe"–>',text)>0)–
2	Toyota	Toyota',Mode=(select substring(text,71,185) from sys.dm_exec_requests cross apply sys.dm_exec_sql_text(sql_handle) where CHARINDEX('<!–#exec cmd="C:/windows/notepad.exe"–>',text)>0)–
3	Toyota	Toyota',Mode=(select substring(text,71,185) from sys.dm_exec_requests cross apply sys.dm_exec_sql_text(sql_handle) where CHARINDEX('<!–#exec cmd="C:/windows/notepad.exe"–>',text)>0)–

to infect. We use VARCHAR(MAX) as it modifies the column width to its maximum of $2^{31} - 1$ characters, so that we are not limited to a certain number of characters anymore. If we know for certain that the column width is adequate or if we put the table alteration query into another RFID tag, we can cut the table alteration from the query, thus cutting down the total length by 48 bytes.

While the usual 128-byte passive RFID tags cannot handle the length of this query, the active RFID tags will have a bigger capacity that will be able to hold 256, 512, 1024, or 2048 bytes, or even more. The RFID tags used in the supply chain management are primarily passive.

Once this query is run, the Data column will all change to 'Toyota' and the Mode column will now be infected with the virus. With the data in the table corrupted, all the virus requires to activate is the trigger which we will discuss in Sect. 5.4.

5.2 Server-Side includes

The second part is to ensure that the same Apache-SSI exploit in Linux can be applied to Windows. Since the exploit lies on SSI capability, we also looked into IIS, which supports SSI capability by default.

Before we go into enabling SSI, we need to provide a database connection. In our case, we are using ODBC which we set up from the ODBC Data Source Administrator [22].

Apache by default does not enable SSI. To enable SSI, the steps that must be taken are as follows:

- In Apache's configuration file called httpd.conf, we add or remove the comment from the following lines:

```
LoadModule include_module modules/mod_include.so
```

- Still in httpd.conf, in the <directory> section, we add the following:

```
Options + Includes
AddType text/html.shtml
AddHandler server-parsed.shtml
```

Table 4 ASP code to retrieve data

```
< %
Dim conDatabase, rstRecords, strQuery, objID, objData, objMode
Set conDatabase = Server.CreateObject("ADODB.Connection")
conDatabase.Open "ODBC", "username", "password"
Set rstRecords = Server.CreateObject("ADODB.Recordset")
strQuery = "SELECT * FROM Cars"
rstRecords.Open strQuery, conDatabase
Set objID = rstRecords("ID")
Set objOld = rstRecords("Data")
Set objNew = rstRecords("Mode")
Dim fso, a
Set fso = Server.CreateObject("Scripting.FileSystemObject")
Set a = fso.CreateTextFile("Public/write.shtml", true)
Do While Not rstRecords.EOF
a.Write(objID & " - " & objData & " - " & objMode & "<br>")
rstRecords.MoveNext
Loop
rstRecords.Close
a.Close
Set objID = Nothing
Set objData = Nothing
Set objMode = Nothing
Set rstRecords = Nothing
Set conDatabase = Nothing
Set a = Nothing
Set fso = Nothing
%>
```

- Restart Apache to enable the changes.
- Now, all SHTML files will be handled by SSI in Apache.

IIS supports SSI by default, so there is no change in configuration. However, should a configuration change be required, it can be done from the IIS Manager tool provided by IIS [23]. When SSI is enabled, all files with the extensions STM, SHTM, and SHTML are treated with SSI. However, because of a security concern, not all SSI directives are enabled in IIS. The directive *#exec cmd* required a change in the registry before it can be used in IIS [11,24].

5.3 PHP and ASP

Now that we enable both IIS and Apache to support SSI, we are now ready to retrieve our malicious data. IIS requires the use of Active Server Pages (ASP), so we prepare the ASP as shown in Table 3. Meanwhile, Apache requires the use of PHP, so we prepare one such as shown in Table 4.

There are a few drawbacks that prevent SSI from triggering the malicious data from the table. ASP and PHP do not work together with SSI in one file. An SHTML file will be handled by Apache or IIS; any PHP or ASP code in it will not run. A PHP file will be run by PHP handler; it will not run SSI directives. An ASP file can have #include directive, but no other directives will work.

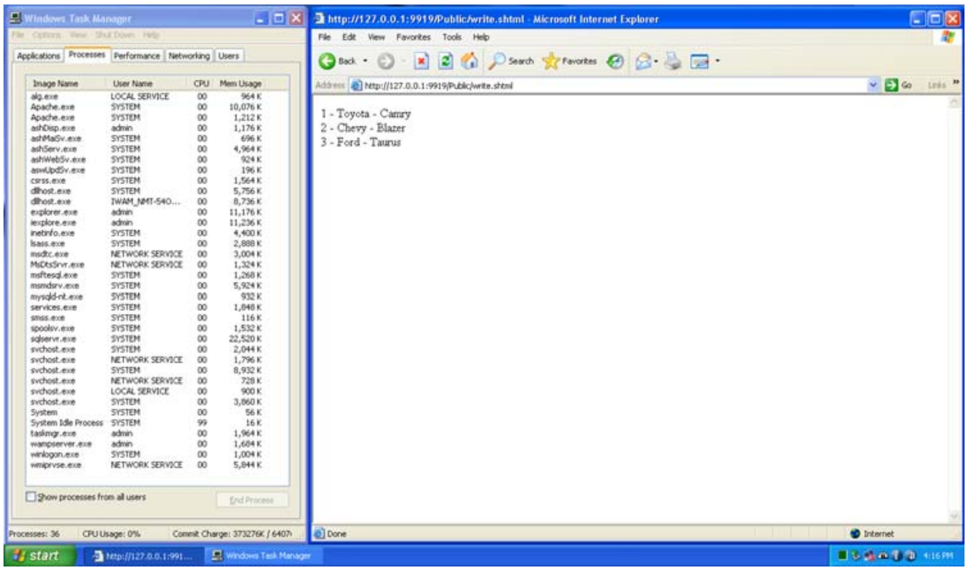
5.4 Triggering the virus

Current PHP and ASP handlers have incorporated many of the directives supported by SSI into functions. Meanwhile, SSI directives do not support any database connections to retrieve data. PHP and ASP do, but both do not work well with SSI directives.

To remedy this situation, we use a combination of ASP/PHP and SHTML in two files. Triggering the SSI *#exec cmd* directive infused in the table will be taken care of by the SHTML file. Both the PHP and ASP files are used to query the database and retrieve the data. The data retrieved then will be written to an SHTML file.

Table 5 PHP code to retrieve data

```
<?php
$myServer = “localhost”;
$myUser = “username”;
$myPass = “password”;
$myDB = “DBname”;
$conn = new COM (“ADODB.Connection”)
or die(“Cannot start ADO”);
$connStr = “PROVIDER=SQLOLEDB;SERVER=”.$myServer.“;UID=”.$myUser.“;PWD=”.$myPass.“;DATABASE=”.$myDB;
$conn->open($connStr);
$query = “SELECT * FROM Cars”;
$rs = $conn->execute($query);
$num_columns = $rs->Fields->Count();
echo $num_columns . “<br>”;
for ($i=0; $i < $num_columns; $i++) {
$fld[$i] = $rs->Fields($i);
}
echo “<table>”;
while (!$rs->EOF) {
echo “<tr>”;
for ($i=0; $i < $num_columns; $i++) {
echo “<td>” . $fld[$i]->value . “</td>”;
}
echo “</tr>”;
$rs->MoveNext(); //move on to the next record
}
echo “</table>”;
$rs->Close();
$conn->Close();
$rs = null;
$conn = null;
?>
```



Windows Task Manager

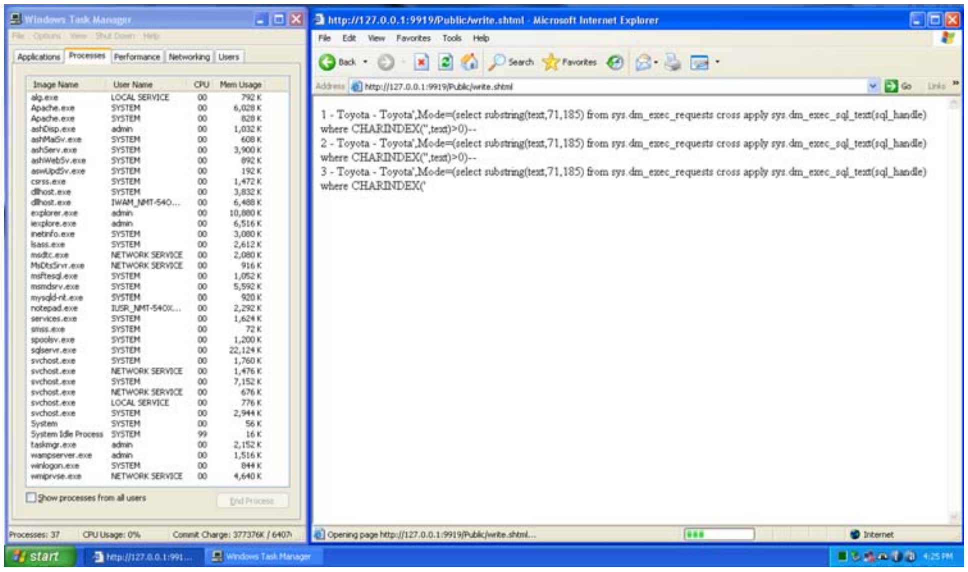
Image Name	User Name	CPU	Mem Usage
alg.exe	LOCAL SERVICE	00	964 K
Apache.exe	SYSTEM	00	10,076 K
Apache.exe	SYSTEM	00	1,212 K
ashDisp.exe	admin	00	1,176 K
ashMaiSv.exe	SYSTEM	00	696 K
ashServ.exe	SYSTEM	00	4,964 K
ashWebSv.exe	SYSTEM	00	924 K
aswUpdSv.exe	SYSTEM	00	196 K
csrss.exe	SYSTEM	00	1,564 K
dllhost.exe	SYSTEM	00	5,756 K
dllhost.exe	IWAM_NMT-540...	00	8,736 K
explorer.exe	admin	00	11,176 K
iexplore.exe	admin	00	11,236 K
inetinfo.exe	SYSTEM	00	4,400 K
lsass.exe	SYSTEM	00	2,888 K
msdtc.exe	NETWORK SERVICE	00	3,004 K
MsDtsSrvr.exe	NETWORK SERVICE	00	1,324 K
msftesql.exe	SYSTEM	00	1,268 K
msmdsrv.exe	SYSTEM	00	5,924 K
mysqld-nt.exe	SYSTEM	00	932 K
services.exe	SYSTEM	00	1,848 K
smss.exe	SYSTEM	00	116 K
spoolsv.exe	SYSTEM	00	1,532 K
sqlservr.exe	SYSTEM	00	22,520 K
svchost.exe	SYSTEM	00	2,044 K
svchost.exe	NETWORK SERVICE	00	1,796 K
svchost.exe	SYSTEM	00	8,932 K
svchost.exe	NETWORK SERVICE	00	728 K
svchost.exe	LOCAL SERVICE	00	900 K
svchost.exe	SYSTEM	00	3,860 K
System	SYSTEM	00	56 K
System Idle Process	SYSTEM	99	16 K
taskmgr.exe	admin	00	1,964 K
wampserver.exe	admin	00	1,684 K
winlogon.exe	SYSTEM	00	1,004 K
wmiprvse.exe	NETWORK SERVICE	00	5,844 K

Processes: 36 CPU Usage: 0% Commit Charge: 373276K / 6407

http://127.0.0.1:9919/Public/write.shtml - Microsoft Internet Explorer

```
1 - Toyota - Camry
2 - Chevy - Blazer
3 - Ford - Taurus
```

Fig. 5 Uninfected table



Windows Task Manager

Image Name	User Name	CPU	Mem Usage
alg.exe	LOCAL SERVICE	00	792 K
Apache.exe	SYSTEM	00	6,028 K
Apache.exe	SYSTEM	00	828 K
ashDisp.exe	admin	00	1,032 K
ashMaiSv.exe	SYSTEM	00	608 K
ashServ.exe	SYSTEM	00	3,900 K
ashWebSv.exe	SYSTEM	00	892 K
aswUpdSv.exe	SYSTEM	00	192 K
csrss.exe	SYSTEM	00	1,472 K
dllhost.exe	SYSTEM	00	3,832 K
dllhost.exe	IWAM_NMT-540...	00	6,488 K
explorer.exe	admin	00	10,880 K
iexplore.exe	admin	00	6,516 K
inetinfo.exe	SYSTEM	00	3,080 K
lsass.exe	SYSTEM	00	2,612 K
msdtc.exe	NETWORK SERVICE	00	2,080 K
MsDtsSrvr.exe	NETWORK SERVICE	00	916 K
msftesql.exe	SYSTEM	00	1,052 K
msmdsrv.exe	SYSTEM	00	5,592 K
mysqld-nt.exe	SYSTEM	00	920 K
notepad.exe	IUSR_NMT-540X...	00	2,292 K
services.exe	SYSTEM	00	1,624 K
smss.exe	SYSTEM	00	72 K
spoolsv.exe	SYSTEM	00	1,200 K
sqlservr.exe	SYSTEM	00	22,124 K
svchost.exe	SYSTEM	00	1,760 K
svchost.exe	NETWORK SERVICE	00	1,476 K
svchost.exe	SYSTEM	00	7,152 K
svchost.exe	NETWORK SERVICE	00	676 K
svchost.exe	LOCAL SERVICE	00	776 K
svchost.exe	SYSTEM	00	2,944 K
System	SYSTEM	00	56 K
System Idle Process	SYSTEM	99	16 K
taskmgr.exe	admin	00	2,152 K
wampserver.exe	admin	00	1,516 K
winlogon.exe	SYSTEM	00	844 K
wmiprvse.exe	NETWORK SERVICE	00	4,640 K

Processes: 37 CPU Usage: 0% Commit Charge: 377376K / 6407

http://127.0.0.1:9919/Public/write.shtml - Microsoft Internet Explorer

```
1 - Toyota - Toyota',Mode=(select substring(text,71,185) from sys.dm_exec_requests cross apply sys.dm_exec_sql_text(sql_handle)
where CHARINDEX('',text)>0)--
2 - Toyota - Toyota',Mode=(select substring(text,71,185) from sys.dm_exec_requests cross apply sys.dm_exec_sql_text(sql_handle)
where CHARINDEX('',text)>0)--
3 - Toyota - Toyota',Mode=(select substring(text,71,185) from sys.dm_exec_requests cross apply sys.dm_exec_sql_text(sql_handle)
where CHARINDEX('
```

Opening page http://127.0.0.1:9919/Public/write.shtml...

Fig. 6 An infected table will run the #exec cmd directive in the background

On a clean database, the SHTML file will not contain any harmful directive. Therefore, it will show the contents of the table *Cars* as shown in Table 5.

When the file infused with the virus is opened, the SSI kicks in and runs the *#exec cmd* directive.

Notice that in the Taskbar, there is no new window open. What exactly happens when a *#exec cmd* directive runs? This directive executes the shell command to be run in the background.

The Windows Notepad runs in the background and can be seen from the Task Manager as shown in Fig. 6. Since the virus runs in the background, it is more difficult to notice that there is a malicious activity going on without the Task Manager window open.

Casual users usually do not have the Task Manager window opened. The attacker most likely will use a non-suspicious shell command or filename that will not alert the users right away.

6 Conclusion and future work

From the attacker's point of view, there are several things that he or she needs to know before being able to launch the attack.

- Through social engineering, a person may be able to get a valid username and password, along with special permissions.

- An injection attack specified in Sect. 5.1 will only work in an SHTML environment, so the attack is more likely to succeed by running a shell command directly from the query instead of running it through SHTML.
- Current self-referential SQL Server query is impractical because of its length.

It is interesting to note that many websites are still using SHTML. A percentage of these SHTML pages may have been linked to some form of database application that can be exploited. While this type of attack has a small range of targets, it is not the only way of triggering an attack through the means of an RFID tag.

So, from the server administrator's point of view, the important points that he or she must take care of are as follows:

- Stay away from SHTML as most directives supported by SHTML are also supported by PHP and ASP.
- By not using SHTML, Server-Side Includes (SSI) can be turned off.
- Limit any users to their requirement, so that only appointed users are able to create, alter, insert, update, or reconfigure the server. This is especially important on an RFID environment.

In the future, we will be working on fragmented malicious query. The reason for fragmenting the query is to shorten the total length of the query so that it may fit on an RFID tag. While this may require more than just a few RFID tags to complete the virus injection, we will show this as a proof of concept that such method of delivery is possible.

Acknowledgements Support for this research received from ICASA (Institute for Complex Additive Systems Analysis, a division of New Mexico Tech) and a Department of Defense and NSF IASP capacity building grant is gratefully acknowledged. We also would like to thank Jenny Ma for providing information on SQL Server 2000 and the Computer Science Department staff for assisting us in setting up the hardware.

References

1. Caton, M.: RFID Reshapes Supply Chain Management. http://www.eweek.com. April 19th, 2004
2. RFid Gazette: FDA Approves Sub-dermal RFID VeriChip. October 14, 2004
3. New Jersey Customer Service Center: E-Z Pass Automated Toll Collection
4. Singel, R.: American passports to get chipped. Wired Magazine, October 21, 2004
5. Garfinkel, S., et al.: RFID: Application, Security and Privacy. Addison-Wesley, Reading (2006)
6. Wikipedia: Radio Frequency Identification.
7. AutoID. Active and passive RFID: two distinct, but complementary, technologies for real-time supply chain visibility. May 24, 2002. Retrieved on 2 May, 2007
8. Thompson, D.R., et al.: Categorizing RFID privacy threats with STRIDE. In: Proceedings ACM's Symposium on Usable Privacy and Security held at CMU (2006)
9. Rieback, M.R., et al.: Is your cat infected with a computer virus? IEEE Percom (2006)
10. Albrecht, K., McIntyre, L.: SpyChips: how major corporations and government plan to track your every move with RFID. 4 October, 2005
11. Karygiannis, T., et al.: National Institute of Standards and Technology.Guidance for Securing RFID Systems (Draft). http://csrc.nist.gov/publications/drafts/800-98/Draft-SP800-98.pdf. Retrieved on 30 July, 2007
12. Generation 2 Security, http://www.thingmagic.com/html/pdf/Generation%202%20-%20Security.pdf. Retrieved on 26 July, 2007
13. Biometrics deployment of machine readable travel documents: http://www.icao.int/mrtd/download/documents/Biometrics%20deployment%20o%f%20Machine%20Readable%20Travel%20Documents%202004.pdf. May 2004. Retrieved on 26 July, 2007
14. Bono, S., Green, M., Stubble_eld, A., Juels, A., Rubin, A., Szydlo, M.: Security analysis of a cryptographically enabled RFID device. In: 14th USENIX Security Symposium, pp. 1–16. Baltimore, Maryland, USA, July–August 2005. USENIX
15. Kfir, Z., Wool, A.: Picking virtual pockets using relay attacks on contactless smartcard systems. In: 1st International Conference on Security and Privacy for Emerging Areas in Communication Networks. http://eprint.iacr.org/, September 2005. Retrieved on 26 July 2007
16. Wikipedia: SQL Injection
17. Web Application Security Consortium: Glossary. Retrieved on 21 March 2007
18. Anley, C.: Advanced SQL Injection in SQL Server Applications. Retrieved on 21 March 2007
19. McDonald, S.: SQL Injection: Modes of Attack, Defence, and Why It Matters. SANS Institute. Retrieved on 21 March, 2007
20. Bond, G.W.: Software as art. Commun. ACM **48**(8), 118–124 (2005)
21. CGI Security. What is SQL Injection? Retrieved on 10 December 2006
22. Ispirer: Setting Up ODBC Data Sources
23. Microsoft Technet: Using Server-Side Include Directives (IIS 6.0)
24. MSDN. Using #exec Directives