.NET Server Security:
Architecture and Policy Vulnerabilities

The unofficial supplement to
Windows .NET Server Security Handbook

Presented at Defcon 10
August 3, 2002
Las Vegas, Nevada

Dr. Cyrus Peikari
Seth Fogie

Overview

Windows .NET Server is Microsoft's new contender against Linux in the server market. Scheduled for release in 2003, .NET Server (which was originally released for beta testing under the codename "Whistler") is re-engineered from the Windows 2000 Server codebase. As Bill Gates himself implied in his “Trustworthy Computing” memo, .NET Server's survival will probably depend on how users perceive its security. In fact, Microsoft has repeatedly delayed the release of .NET Server, citing security reasons. Thus, this speech will focus on the new security features in .NET Server -- and potential attacks against them. The purpose is to identify early vulnerabilities while the OS is still in beta in order to help Microsoft safely release the product. In addition, we address Microsoft corporate security policies that might negatively impact the perception of .NET Server security.

Contents:

Introduction

Windows .NET Server is Microsoft's new contender against Linux in the server market. Originally scheduled for release in 2001, .NET Server has been delayed several times, most recently for “security reasons.” Consider the following timeline:

.NET Server Pre-release history

- Code has currently been in Beta for over a year
- Original Codename: Whistler
- Original Expected Release: Late 2001
- Original Release Candidate Name: Windows 2002 Server
- Trustworthy Computing Initiative release rollback: Mid 2002
- Final name: Windows .NET Server
- Currently Expected out: Mid 2003 (Over 2 years of Beta testing)

.NET Server (which was originally released for beta testing under the codename "Whistler") is the base enterprise operating system that is re-engineered from the Windows 2000 Server codebase. .NET Server's survival will probably depend on how users perceive its security. This paper will focus on some of the new security features in .NET Server -- and how to break them. The purpose is to identify early vulnerabilities while the OS is still in beta in order to help Microsoft release a safer product. In addition, we address Microsoft corporate security policies that might negatively impact the perception of .NET Server security.

Disclaimer

This paper is not meant to criticize anyone, least of all the hard-working men and women who program for Microsoft. We respectfully offer these suggestions in the hope that they will be useful to all who work towards security.

Windows Product Activation

Windows Product Activation (WPA) currently ships with the release candidate of .NET Server. WPA is a controversial anti-piracy licensing scheme that was introduced in Windows XP. Retail copies of .NET Server, as well as some copies that are preloaded on OEM-purchased servers, will now require activation via the Internet or by telephone. (volume licensing does not require activation).

Critics have reported several potential problems with WPA. In fact, if Microsoft does not withdraw WPA altogether, .NET Server is likely to be a total market disaster. Rather than suffer such humiliation, the preponderance of administrators who are still loyal to Microsoft will defect *en masse* to Linux.

Privacy and the BSA

As others have pointed out, WPA is also unlikely to be effective as an anti-piracy tool. For example, hackers cracked WPA as soon as the first beta was released. Thus, WPA is potentially more useful as a means to scare legitimate consumers and businesses into keeping up with vendor license demands. However, when viewed in the historical context of Microsoft's oppressive intellectual property persecution, the WPA raises privacy issues of Orwellian proportions.

This history centers on an organization known as the *Business Software Alliance (BSA).* Despite the impression they have endeavored to attach to their acronym, the BSA, unlike the FBI or CIA, has no law enforcement or government affiliation. They have been described as purely for-profit "hired muscle" used by major software companies such as Microsoft and others to intimidate honest companies.

As an example, the BSA regularly targets one city at a time with "strong arm" tactics. They will send an official-looking letter, which reads like an ultimatum, to various tech companies in the target city. The purpose is to trick honest companies in to voluntarily submitting to a software audit. Frequently, the ruse works. The unfortunate company, struck with panic over a full BSA "investigation," self-reports that it cannot find the documentation for all of its software licenses and ends up paying huge sums to the BSA. In fact, the Association of Chartered Certified Accounts in the UK described the mailing as "heavy-handed and 'questionable.'" It is also very profitable. According to the BSA, they have collected over $75 million in such "fines" over the last decade

This city-wide BSA mailing is typically followed by a persistent radio campaign stating that the BSA has "declared war" on the city and warning companies to submit to a voluntary "inspection." In the same commercial they seem to coax disgruntled ex-employees to retaliate against their hated ex-boss by reporting them for software license violations – which (incredibly) may be all it takes for the BSA to obtain a police escort and search warrant.

After you are "audited", the BSA will assess a "fine" of up to several hundred thousand dollars. The curious part is that most companies are so scared and bewildered by the whole BSA spectacle that they gladly pay without a word of protest.

Thus, WPA could potentially be abused as a tool for targeting honest individuals and companies. After you install .NET Server, Microsoft suddenly has detailed personal information about your system and IP address. From there it could theoretically be hours before the BSA turns up at your door with the local police and a warrant to conduct a search, although there is no evidence that such a policy exists.

Unfortunately, the BSA often dupes honest law enforcement officers into seeking a warrant. Because BSA “officials” wear suits and ties, and use official-sounding terms, they invariably impress local police enough to get them scrambling. Even when presented with the most flimsy pretext of evidence, the poor law enforcement officer is so overwhelmed by the BSA suits that he feels compelled to go along with the charade.

Technical Problems with WPA

Rob Robinson relates this anecdote of his experiences with WPA:

“WPA is unquestionably one of the worst abominations that MS has imposed on its customers. The latest problem arose when we attempted to use TS (Terminal Services) license server. The server is not required for "one user" administrative access. This is the mode in which we normally use TS knowing about the hassles of client licenses. Unfortunately, an "upgrade" of the server version of WXP (which we will call WNES for lack of a better name) to build 3604 resulted in TS being disabled until the license server is installed and activated. Installing this server involves transmitting mandatory, detailed user information to MS and an online activation. We did this and then encountered the fun step.

The system now wants you to enter a 5x5 matrix of alphanumeric characters for each client license. These characters are supposedly obtained not on-line, but by human to human telephone interaction with MS. Yesterday, I spent about 45 minutes on the telephone with MSDN customer support and technical support. No one with whom I spoke had the foggiest idea of how one installs TS client licenses and stated that no information was available in their MS database. Another series of phone calls to MS today resulted in our being told that no one knew how to do it and the only solution was to format the hard drive, re-install everything and thereby gain another 90 days of TS usage. In the interim, we lost our TS access.

I finally obtained the correct TS client licensing telephone number from MSDN sales. I called it today. The person first said that he didn't know anything about MSDN client licensing and then decided that, yes, this was his job. The next difficulty was that the DSL connect happened to be down. No problem, he said, lets do it by telephone. Selecting that choice did, for the first time, display a MS telephone number for the client licensing. He now checked to see if our licensing server had been activated. The computer said yes and the database said no. The MS employee then decided that the server activation was stored in a (separate) MSDN database. He "cut and pasted" the 35 alphanumeric character code for our system from one database to another and all the computers now agreed that there was a working and activated license server. The next step was to enter the 5x7 character license pack code. This was achieved after some fun telephone interaction with the phonetic alphabet. Our server now said that all was well. We had an activated TS license server and 10 licenses. Unfortunately, Remote Desktop still thought that there was no proper licensing. No amount of license server starts, stops, re-starts or other steps including a system cold start would convince the OS *that the licensing existed. We finally removed TS licensing from the system. The only good news is that it was then possible to again have administrative Remote Desktop access.*

Yes, I do know that this is beta code. Nevertheless, the TS activation/licensing procedure is incredibly cumbersome and there are obviously few MS support people that understand it. Beta or not, it is impossible to use TS if the licensing procedure glitches prohibit this functionality. The experience underlines why this type of activation/licensing is such a bad scheme. To make matters worse, I was told that it is necessary to repeat the whole process if one does a re-install of the OS."

What does WPA report?

The software company "Fully Licensed" completely reverse-engineered the WPA process. Remembering the ignominy that befell dear Dimitry Sklyarov at Defcon last year, because of DMCA restrictions we cannot reprint the excellent analysis by Fully Licensed.

However, based on their findings, Fred Langa reported in *Information Week* that when you register XP software the OS sends Microsoft a unique 50-digit hash of your machine. In addition to the software license number, this fingerprint is derived from the following aspects of your system:

1. CPU serial number
2. CPU model number/type
3. Amount of RAM in the system
4. Graphics adapter hardware ID string
5. Hard drive hardware ID string
6. SCSI host hardware ID string (if present)
7. Integrated development environment controller hardware ID string
8. MAC address of your network adapter
9. CD-ROM drive hardware identification string
10. Whether the system is a dockable unit (e.g. a notebook) or not

Worse, even after the product has been fully registered, WPA "phones home" to Microsoft from time to time. Thus, if you have changed your system components, or if there are problems with the Microsoft central database, your system locks into a reduced-functionality mode. It is unclear what other investigation or action Microsoft takes against you after that.

Kerberos Authentication

Microsoft's implementation of Kerberos

In Windows .NET Server, Microsoft's implementation of Kerberos v5 is the default network protocol for authentication within a domain. The Kerberos v5 protocol verifies both the identity of the user and network services. This dual verification is known as *mutual authentication*.

The Kerberos protocol was initially developed in the 1980s at the Massachusetts Institute of Technology under a project known as "Athena." The name *Kerberos* (*Cerberus* in Latin) comes from the mythical three-headed dog that guards the entrance to Hades. The goal of the project was to design Authentication, Authorization and Auditing services (all three heads of Kerberos). However, they only implemented Authentication.

Microsoft's implementation of Kerberos includes all three heads, Authentication, Authorization and Auditing. Kerberos provides strong authentication methods for client/server applications in distributed environments by taking advantage of shared secret key cryptography and multiple validation technologies.

By migrating to Kerberos in Windows 2000 Server and .NET Server, Microsoft demonstrated another admirable move towards industry standards. This chapter we will introduce you to the components that comprise Kerberos under Windows .NET Server, in addition to the authentication process. In addition, we will point out known attacks against Kerberos in a Windows environment.

Kerberos Authentication

Kerberos runs on a system of *tickets* issued by the Key Distribution Center (KDC). To gain access to a network resource, you must have a ticket for authentication. The KDC is the main communication intermediary in this scheme and runs as a service on Windows .NET domains. In fact, every Windows .NET domain controller is a KDC by default. The purpose of the KDC is to grant initial tickets and Ticket-Granting Tickets (TGTs) to *principals*. In Kerberos, a principal can be a user, a machine, a service, or an application. By presenting a pre-shared secret, each principal gets a unique TGT.

The KDC is comprised of two components, which are the Authentication Service (AS) and the Ticket-Granting Service (TGS). The AS is the first subprotocol activated when the user logs on to the network. The AS provides the user with a logon, a temporary session (encryption) key, and a TGT. The TGS is unique for each session and is only issued to principals who can present a valid TGT.

When a principal wants to communicate with another principal, it presents its unique TGT to the KDC. Fig. 7-1 shows an overview of the Kerberos communication sequence.

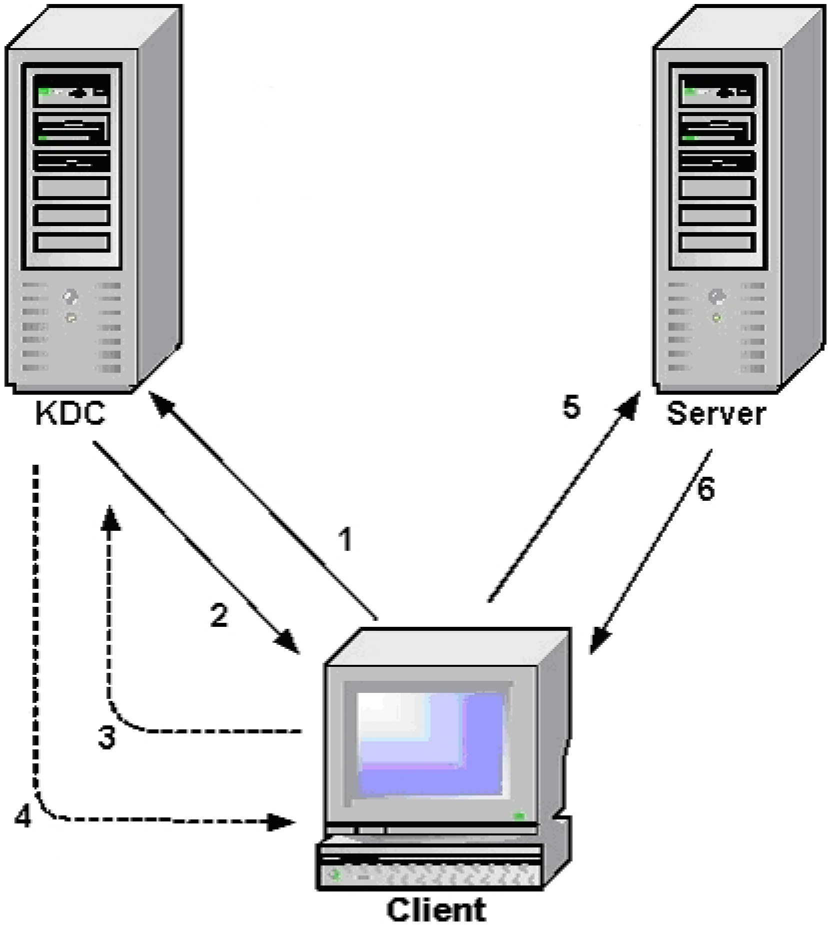

Figure 7-1

As shown in the figure, authentication executes this sequential process, as follows:

1. The principal (in this example, the Client) first makes an authentication service request to the KDC for a Ticket Granting Ticket.
2. The KDC responds to the Client with a Ticket Granting Ticket (TGT). This includes a key (ticket session key) and is encrypted with the Client's password.
3. The Client now uses its new TGT to request a Ticket Granting Service (TGS) ticket in order to access the other principal (in this example, the Server).
4. The KDC responds to the Client by issuing a Ticket Granting Service (TGS) ticket to the client to access a specific resource on the Server.
5. The Client now presents the TGS as a request to the Server
6. The Server authenticates the Client by acknowledging the TGS. If mutual authentication is specified, then the Client reciprocates by authenticating the Server as well.

Accessing cross-domain network resources

In the Kerberos protocol, a *realm* is the equivalent of a Windows .NET domain. Using the example above, suppose the Client would like to access resources from an entirely different domain. As you recall, the Client first received the TGT from the KDC in its own domain (Domain 1). However, this TGT only works in the current domain (Domain 1). If the Client wants to access a resource in a trusted domain (Domain 2), it must request a new TGT. Thus, the KDC from Domain 1 issues the Client a new TGT that will provide authentication to the KDC in Domain 2.

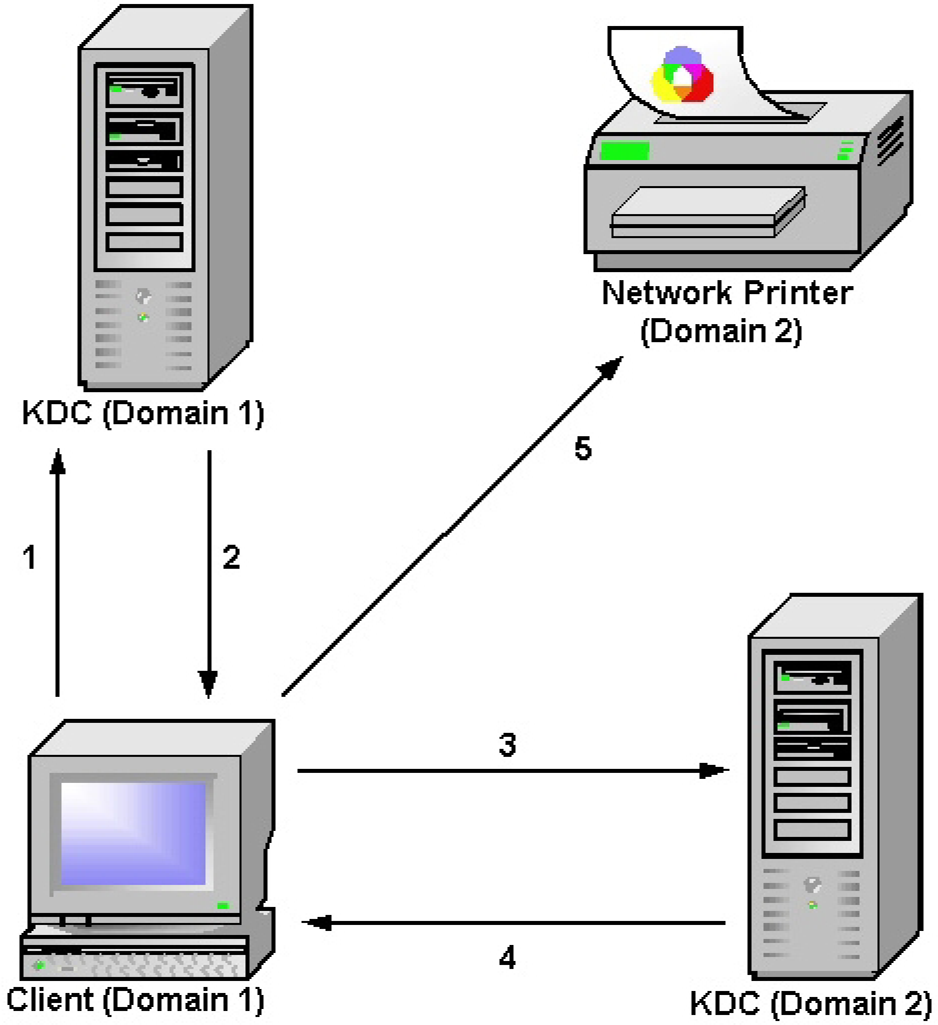

Figure 7-2

The steps involved in Kerberos cross-domain authentication are as follows:

1. The Client on Domain 1 wishes to access a network resource (in our example, a network printer) on remote Domain 2. The Client has already been authenticated to the KDC on Domain 1 and has received a TGT. The Client now presents the TGT to the KDC on Domain 1 and requests a TGS to access the remote network resource.

2. The KDC on Domain 1 cannot provide a TGS to the network resource on Domain 2 since the network resource is on a remote domain. Instead, the KDC on Domain 1 responds to the Client with a TGT for Domain 2.
3. The client presents this new TGT to the KDC on Domain 2.
4. The KDC on Domain 2 responds with a TGS for the network resource.
5. The Client accesses the network resource on Domain 2 using the new TGS.

Notes:

- Active Directory (AD) now supports cross-forest Kerberos transitive trusts

- Authentication on the front-end Web server can be transitioned to Kerberos in the backend.

- You can configure to which backend servers delegation is made.

- Front-end servers are part of a trusted KDC that can handle authorization and user logon without credentials.

Weaknesses in the Kerberos Protocol

While Kerberos is a drastic improvement in security over the archaic NTLM (NT LAN Manager), Kerberos as implemented in Windows has still been shown to be potentially vulnerable. For example, Frank O'Dwyer provides the following attack:

“It is well known that the LM and NTLM authentication schemes used by NT4 (and for backwards compatibility in Windows 2000) are very susceptible to offline password guessing attacks. This has been ably demonstrated by password-cracking tools such as l0phtcrack. However, the question of whether it is feasible to adapt these techniques to attack the Kerberos 5 authentication scheme used by Windows 2000 does not appear to have received the same level of public attention. It is also worrying that the general presumption seems to be that Kerberos 5 solves the password cracking issue once and for all, provided that Kerberos alone is used in a domain.

In fact Kerberos 5 has long been known to have vulnerabilities to offline password guessing attacks. This paper investigates the feasibility of exploiting one of these vulnerabilities to build a point-and-click 'l0phtcrack'-style password cracking tool for Windows 2000 Kerberos. We do not go as far as actually building this tool, but consider what would be involved in making one, and how well and how fast it might work in recovering passwords.

N.B.: Password-based login is not the only option in Kerberos 5, nor is it the only option in Windows 2000. It is also possible to login using a public key based scheme, PKINIT, that does not suffer from the problem outlined here. Windows 2000 includes support for this scheme too, with or without smartcard assistance. This entire discussion applies only to the option which is enabled by default, and which is most widely used, which is to use passwords to login. If PKI is not an option for you, there are other potential defenses and fixes that are also suggested in the conclusions section of the paper.

This problem is also explicitly stated in RFC1510:

"Password guessing" attacks are not solved by Kerberos. If a user chooses a poor password, it is possible for an attacker to successfully mount an offline dictionary attack by repeatedly attempting to decrypt, with successive entries from a dictionary, messages obtained which are encrypted under a key derived from the user's password. [RFC1510]

Vulnerability

In order to mount an offline dictionary or brute force attack, some data that can be used to verify the user's password is needed. One way to obtain this from Kerberos 5 is to capture a login exchange by sniffing network traffic.

In Kerberos 5 a login request contains pre-authentication data that is used by the Kerberos AS to verify the user's credentials before issuing a TGT. The basic pre-authentication scheme that is used by Windows 2000 and other Kerberos implementations contains an encrypted timestamp and a cryptographic checksum, both using a key derived from the user's password.

The timestamp in the pre-authentication data is ASCII-encoded prior to encryption, and is of the form YYYYMMDDHHMMSSZ (e.g. "20020304202823Z"). This provides a structured plaintext that can be used to verify a password attempt - if the decryption result "looks like" a timestamp, then the password attempt is almost certainly correct. A password attempt that recovers a plausible timestamp can also be verified by computing the cryptographic checksum and comparing it to that in the pre-authentication data.

Obtaining the password verification material

Using a test Windows 2000 domain, a login attempt for the user 'frank' with the password 'frank' was made and the exchange was captured using the freely available sniffing tool WinDump (a windows implementation of tcpdump). This was then investigated using the freely available ASN.1 decoder dumpasn1 and the Kerberos V5 specification.

As expected, the capture contained the following pre-authentication data:

```
  2 30   72:   SEQUENCE {
```

```
 4 A1    3:    [1] {
 6 02    1:       INTEGER 2
         :      }
 9 A2   65:     [2] {
11 04   63:       OCTET STRING, encapsulates {
13 30   61:           SEQUENCE {
15 A0    3:             [0] {
17 02    1:                INTEGER 23
         :              }
20 A2   54:             [2] {
22 04   52:                OCTET STRING
         :              F4 08 5B A4 58 B7 33 D8 09 2E 6B 34 8E 3E 39 90
         :              03 4A CF C7 0A FB A5 42 69 0B 8B C9 12 FC D7 FE
         :              D6 A8 48 49 3A 3F F0 D7 AF 64 1A 26 3B 71 DC C7
         :              29 02 99 5D
         :              }
:           }

:         }

:     }

:   }
```

The second OCTET STRING contains the encrypted timestamp that can be used to seed an offline attack. The etype 23 appears to be a Microsoft specific
etype, based on RC4-HMAC. The details of this are publicly documented in the Internet Draft draft-brezak-win2k-krb-rc4-hmac-03.txt.

Decrypting the timestamp

The Brezak Internet draft also contains a detailed description of how the RC4 key is derived from the user's password, as well as pseudo-code for decrypting and verifying the timestamp. Implementing this is straightforward (the code here used the OpenSSL cryptographic libraries) and yields the necessary password test function for mounting an offline attack.

As mentioned above, it is not necessary to compute the expensive embedded cryptographic checksum in order to verify a password - one can simply decrypt and look for an ASCII string that looks like a timestamp. If the decryption does not recover a timestamp, the password tried is incorrect. If the decryption does recover a timestamp, the password tried is almost certainly correct, and if desired the cryptographic checksum in the encrypted data can be used to further verify this. As most passwords tried will be incorrect, the extra overhead involved in doing this extra verification after the initial check for a recovered timestamp succeeds is minimal.

Simulating an attack

Given the above password test function, and some captured pre-authentication data to verify passwords against, then implementing a password cracker is straightforward. We have not done this, but the obvious approach is to use existing well-known techniques (e.g. as used by Alec Muffet's Crack program, and l0phtcrack), and indeed it is easy to adapt existing code by replacing the password testing component. It is not much of an additional step to automatically capture the pre-authentication data, resulting in a point and click 'script kiddie' tool.

The code given here does not go as far as implementing a full-blown password cracker of any kind. Instead it implements the password test function from the Brezak Internet draft, and iterates it in a simulation of 1,000,000 brute force trials against the example pre-authentication capture shown above. This yields timing information which can be used to estimate the efficiency of a real world attack program.

It would be straightforward to implement a "point and click" Windows 2000 Kerberos cracking tool that would require minimal knowledge on the part of the user, that would be widely deployed, and that furthermore would not require administrative access to a domain controller or indeed to any machine on the target network.

Such a tool can easily be assembled from publicly available code and specifications, and could automatically sniff exchanges between domain controllers and users in order to harvest weak Windows 2000 passwords even in a pure Windows 2000 domain. The same tool could target logins directed at other Kerberos implementations as well, by implementing additional cryptographic methods.

We also conclude that such a tool would be highly effective against dictionary-derived passwords, short passwords (<9 characters, depending on entropy and character set), and/or passwords drawn from a restricted character set."

Hacking PKI

The .NET Server security architecture supports PKI. Although the weaknesses of PKI and smart cards (next section) have been well described and are not limited to .NET Server, we will briefly treat the subject here for completeness. PKI provides a strong framework for authentication, but like any technology it is vulnerable to hackers. It is a mistake to think that PKI is a panacea. As always, it is important to combine PKI with other layers of defense in your security policy. In this section we will review some of the ways that PKI can be hacked.

An example of a vulnerability in one implementation of PKI occurred in mid-March 2001. Verisign informed Microsoft that two Verisign digital certificates had been compromised by social engineering, and that they posed a spoofing vulnerability. In this case, VeriSign had issued Class 3 code-signing digital certificates to an individual who fraudulently claimed to be a Microsoft employee. Because the certificates were issued with the name "Microsoft Corporation", an attacker would be able to sign executable content using keys that prove it to be from a trusted Microsoft source. For example, the patch you thought was signed by Microsoft could really be a virus signed by with the hacker's fraudulent certificate.

Such certificates could also be used to sign ActiveX controls, Office macros, and other executable content. ActiveX controls and Office macros are particularly dangerous, since they can be delivered either though html-enabled email or directly through a web page. The scripts could cause harm automatically, since a script can automatically open Word documents and ActiveX controls unless the user has implemented safeguards.

Iin situations like this the bogus certificates should be have been placed immediately on a Certificate Revocation List (CRL). However, VeriSign's code-signing certificates did not specify a CRL Distribution Point (CDP), so a client would not be able to find and use the VeriSign CRL. Because of this, Microsoft issued a patch that included a CRL containing the two certificates. In addition, the Microsoft patch allowed clients to use a CRL on the local machine, instead of a CDP.

In addition to this example, several observers have pointed out potential weaknesses in PKI. For example, Richard Forno, former Chief Security Officer at Network Associates, has shown how incomplete PKI implementations can give online shoppers a false sense of security. According to Forno, while PKI ensures that the customer's initial transmission of information along the Internet is encrypted, the data may subsequently be decrypted and stored in clear text on the vendor's server. Thus a hacker can bypass the strength of PKI if he can access the clear text database. In fact, rogue employees could easily sniff the data as it travels on the wire from within the corporate network.

Thus, when implementing PKI it is important to consider network security from a holistic perspective. Fred Cohen sketched a list of potential vulnerabilies in his seminal paper titled "50 Ways to Defeat PKI." Most of these attacks involve basic social engineering, Denial of Service, or cryptographic weakness exploitation. Nevertheless, when takes as a whole this list demonstrates that PKI is not infallible.

Smart Card Hacking

A *smart card* typically describes a plastic strip the size of a credit card that has an embedded microprocessor. By taking advantage of public-key infrastructure (PKI), smart cards can simplify solutions such as interactive logon, client authentication, and remote logon. The use of smart cards is growing at an accelerating rate.

However, like any technology, smart cards are vulnerable to attack. In addition to the inherent weaknesses of PKI described above, smart cards may be vulnerable to physical attacks. This section will review smart card technology and will show a brief sample of attacks against them. By understanding these vulnerabilities, you can make an informed decision on whether to utilize .NET Server's streamlined support for smart cards.

Smart Card Advantages

Advantages that smart cards provide include:

- Tamper-resistant and permanent storage of private keys

- Physical isolation of secure private key computations from other parts of the system

- Ease of use and portability of credentials for mobile clients

One advantage of smart cards is that they use PINs instead of passwords. PINs do not have to follow the same rules as "strong passwords" because the cards are less susceptible to brute-force dictionary attacks. A short PIN is secure because an uncompromised smart card will lock when too many wrong PIN inputs are attempted in a row. Furthermore, the PIN itself is never transmitted over-the-network, so it is protected from classic sniffing attacks.

Unlike a password policy, it is not necessary to change the PIN frequently. In fact, there is no change PIN functionality available through the standard desktop logon interface as there is for passwords. This is because the change PIN capability is only exposed to the user when a private key operation is being performed. This is due to the lack of standards for how PINs are managed across card operating systems, thus preventing PIN management from being done at the operating system layer.

Hacking Smart Cards

In 1998, an extensive and well-organized phone-card piracy scam demonstrated how vital proper encryption can be. As reported in Wired Magazine, Criminals from the Netherlands flooded Germany with millions of illegally recharged telephone debit cards. The cards, designed for Deutsche Telekom payphones, use a simple EEPROM chip, developed by Siemens Corp. that deducted value from the card as minutes were used up.

Ordinarily, once the credit balance reached zero, the cards would be thrown away or given to collectors. However, the Dutch pirates found a way to bypass the simple security and to recharge the cards without leaving any physical evidence of tampering. The pirates bought up thousands of spent cards in bulk from collectors, recharged them, and resold them at a discount to tobacco shops and other retail outlets across Germany. The damage from this piracy was estimated to amount to $34 million dollars.

Using hardware reverse engineering, pirates could understand the simple encryption stored on the chip. In addition, they found a bug that could allow the stored monetary value to be reset.

Hardware Reverse Engineering

Hardware attacks on smart cards have traditionally required access to sophisticated laboratory equipment. For example, one way to attack smart cards involves the use of an electron microscope. Using careful etching techniques, reverse engineers can physically “peel away” layers of the microprocessor. Next, through image processing they can often get a fair guess at the contents of the memory registers.

One report states that the Intel 80386 was reverse engineered in about two weeks. The output was available in the form of a mask diagram, a circuit diagram or even a list of the library cells from which the chip was constructed.

Jn addition, more sophisticated attacks are possible with the proper equipment. One report published by Sandia National laboratories involved “looking through” the chip. This attack, known as Light-Induced Voltage Alteration, involves probing operating ICs from the backside with an infrared laser to which the silicon substrate is transparent. This non-destructive method induces photocurrents that allow the researcher to probe the device's operation and to identify the logic states of individual transistors. Similarly, Low-Energy Charge Induced Voltage Alteration uses the low- energy electron beam generated by a scanning electron microscope to produce a surface interaction phenomenon that produces a negative charge-polarization wave. This allows the resesarcher to image the chip in order to identify open conductors and voltage levels without causing damage.

EEPROM Trapping

It is often easier to go directly after the EEPROM contents. In EEPROM based devices, erasing the charge stored in the floating gate of a memory cell requires an unusually high voltage, such as 12V instead of the standard 5V. If the attacker can circumvent the high voltage charge, then the information will be trapped.

With early pay-TV smartcards, a dedicated connection from the host interface supplied the programming voltage. This allowed attacks on systems in which cards were enabled for all channels by default, but those channels for which the subscriber did not pay were deactivated by broadcast signals. Thus, you could block the programming voltage contact on the smartcard with tape or by clamping it inside the decoder using a diode. This prevented the broadcast signals from affecting the card. The subscriber could then cancel their subscription without the vendor being able to cancel their service.

Once the contents of the EEPROM are trapped, here are many methods to access the goods. For example, attackers can use any of the following means:

- raising the supply voltage above its design limit
- lowering the supply voltage below its design limit
- resetting random memory locations using ultraviolet light in order to find the bit controlling read-protection
- exploiting weaknesses in the ROM code

- exploiting weaknesses in the EEPROM code

In order to thwart these attacks, some IC chips have sensors that force a reset when voltage or other environmental conditions go out of range. However, this can cause massive performance degradation because of false positives. Imagine if your smartcard went dead every time the power surged during system startup. For this reason, such defenses are difficult to implement.

Power Consumption Analysis

Power consumption analysis involves monitoring a smart cards' power consumption in order to assist in code breaking. A smart card does not have its own power supply; rather, it draws power from the smart card reader when it is inserted. This power is required to run the IC chip – for example, in performing cryptographic calculations.

Using sensitive equipment, it is possible to track differences in smartcard power consumption. This could make it possible to recover a card's secret key. By watching for changes in power consumption, a researcher can obtain clues. This is because the calculations used to scramble the data depend on the values of the secret key.

For instance, one simple attack involves watching an oscilloscope graph the power consumption of a card. The key is processed binary bits that are either zeros or ones. If a chip consumes slightly more power to process a one than a zero, then the key could be extracted simply by reading the peaks and valleys in the graph of power consumption.

A more sophisticated statistical attack known as *differential power analysis* can be used to extract the key even when it is not readily understandable from the power consumption data. This technique allows the researcher to extract each bit of the key by making guesses and testing each several times.

IIS 6.0

Security minded professionals cringe at the thought of using IIS on a public server. IIS is the single most hacked application in the history of software. However, with IIS version 6.0+, Microsoft has made greater efforts toward security.

Out of the box, IIS 6 sports security improvements. Gone are many of the extras that have historically made IIS vulnerable. There is no ASP support, no VBS support, and no scripting support of any kind. Only static htm and html pages and images are supported. In addition, the infamous IUSR account has been severely restricted by creating a new account running as a NetworkService. This is a restricted permission account, which is designed to stop the known abuses of IUSR problems.

The IIS Lockdown Wizard now comes bundled with IIS. This is Microsoft's attempt to introduce a modicum of baseline security. It is especially useful to new administrators who desire a basic level of security, but who do not want to read 15 books, 45 whitepapers, and another 500 articles on how to secure IIS.

A significant security enhancement to IIS 6 is the way it handles processes. All add-ins, modules, and 3rd party applications are placed in their own sandbox. This helps ensures that they cannot interfere with other processes, and it makes IIS 6 more stable than its predecessors. Multiple web sites are also handled this way, which prevents one webmaster from accessing and altering another's web site.

However, because this program is still in beta, it has not been thoroughly tested.
Even at this early stage there have been several reported vulnerabilities in II 6.0.
For example, an Unchecked Buffer in Index Server ISAPI Extension permitted an attacker to insert unchecked code into a server request that would overwrite a buffer in the Index server DLL. Thus, a hacker could overflow the buffer with 240 bytes of data and cause the register to be overwritten with the address location of their choice. If the hacker gained control of this address, she could insert a slide/shell code combination and theoretically get root. Since this weakness ran as the system account, it is one example of a potentially severe weakness.

Remote Desktop / Remote Assistance

Integrated remote control is one of the most useful features to be included in .NET Server. Both Remote Desktop and Remote Assistance provide unique advantages. However, like all useful features, these too can be exploited. For example, we will show how to turn them into a backdoor.

Remote Desktop

This tool obviates PC Anywhere. It allows an authorized remote user to connect to their machine from anywhere, provided a connection exists. To set up this program, the operating system simply has to be told to accept incoming requests for Remote Desktop. If the server admin wants to allow multiple users to connect (one at a time), they can add extra accounts to the remote desktop settings as well. While this information is relatively secure, as is the connection, it is important to remember that the use of Remote Desktop can by abused remotely by brute force attacks and other traditional attacks. In addition, the connection is protected by a user name and password only, which means the entire security of Remote Desktop depends upon the strength and secrecy of that password.

Remote Assistance

Remote assistance is similar to the remote desktop, except it allows two people to be connected to a computer at one time. Typically, this program will be used by a novice that needs the help of a technician. To receive help, the novice selects the remote assistance option from their help page and sends the technician an email, MSN message, or file which allows the technician to connect to the computer. Unlike remote desktop that is protected by a password, remote assistance does not have to be protected by a password. This can cause security problems.

To illustrate, if a novice is asking help of the local network guru, what are the chances they will not included a password? The likelihood is quite high. In the mind of the novice, this isn't a problem since they are sending the message via email. After all, only the technician will receive the message.

Unfortunately, the remote assistance file is nothing more than an encrypted link that is sent as plaintext to the technician. Therefore, any sniffer can see the link and a hacker could recreate the link and connect to the novice's computer instead of the technician. With a little social engineering, the hacker could talk the novice in to given them full control and could install a backdoor or more in a few minutes.

As this scenario illustrates, the use of remote desktop and remote assistance is an excellent opportunity for a hacker. While it may take some technical prowess, exploiting the remote control features of .NET is one security issue that must be considered.

In addition to the obvious security issues, Beta 3 of .NET server's remote assistance program connects to Microsoft's web site, which acts as a middle man between the novice and helper. Since this link must include IP information of the novice's computer, and the web server can detect the IP address of the helper as they connect, one wonders what the purpose of this is. Windows XP does not require the use of an intermediate web site; instead it uses an XML file with the information included in the file. Why the switch? Regardless of the method, either can be sniffed. Is one really any more secure than the other??

Wireless Standard Weaknesses

Wireless technology is the Achilles heel of modern network security. For example, just in Dallas alone, hackers have documented over 800 unsecured wireless Access Points. .NET Server includes support for the 802.1x wireless protocol, but it should be implemented with extreme caution. The latest methods for hacking wireless networks are presented at the annual DallasCon Wireless Security Conference (www.dallascon.com). In this section we will simply explain the well-known weaknesses in WEP.

The *Wireless Equivalent Privacy* (WEP) protocol defines encryption and authentication method is used to secure wireless data. However, the problem with WEP is in the way the data is encrypted. As discovered by researchers, WEP can be cracked by anyone with a sniffer. However, although cracking WEP is possible on the typical home-owned WLAN, it would take two to four weeks to capture enough data to successfully extract the key. In other words, by simply enabling WEP and changing the secret key periodically, a home user can be fairly certain that her WLAN will not be hijacked. However, corporate users with high traffic flux must beware.

(c)The Secret Key

WEP incorporates two main types of protection: a secret key and encryption. The secret key is a simple 5- or 13-character password that is shared between the access point and all wireless network users. This key is used in the encryption process to uniquely scramble each packet of information with a password. To do this, WEP defines a method to create a unique secret key for each packet using the 5- or 13-characters of the pre-shared key and three more psuedo-randomly selected characters picked by the wireless hardware.

For example, let us assume that our pre-shared key was “games”. This word would then be merged with “abc” to create a secret key of “abcgames”, which would be used to encrypt the packet. The next packet would still use “games” but would concatenate it this time with “xyz” to create a new secret key of “xyzgames”. This process would randomly continue during the transmission of data. This changing part of the secret key is called the Initialization Vector because it initializes the encryption process for each packet of data sent.

(c)XOR

It is important to understand the basics of XOR when discussing RC4 and WEP because XOR is used in the encryption process. In short, XOR takes each corresponding bit in a byte and compares them by asking “Is this bit different from that bit?” If the answer is yes, the result is 1; otherwise it is a 0. In addition, just as the resulting bit can be deduced by comparing the first two columns, the same can be said about the original bit if the XOR bit and resulting bit are compared. (for example, 1 XOR 1 = 0 → 0 XOR 1 = 1). This is an important part of how and why WEP is crackable.

(c)RC4

RC4 is the encryption algorithm used to cipher the data sent over the airwaves. RC4 is a very simple and fast method of encryption that scrambles each and every byte of data sent in a packet. It does this through a series of equations using the previously discussed secret key.

RC4 actually consists of two parts: the Key Scheduling Algorithm and the Psuedo Random Generation Algorithm. Each part is responsible for a different part of the encryption process. However, before discussing the algorithm in detail, it is important to understand what an array is.

(c)Array Swapping

An array is a programming term used to hold multiple values. For example, consider the alphabetArray(26). This array would hold 26 values, with each value represented by the number in the array. For example:

alphabetArray(1)=a
alphabetArray(2)=b
alphabetArray(3)=c

However, in the case of a secure application of an array, we want to scramble the values held in each position. If this were not done, a hacker could easily predict what was in alphabetArray(26). To do this, a swapping function can be performed on the array. For example, consider the following illustration:

alphabetArray(1)=A
alphabetArray(2)=B

Swap (alphabetArray(1), alphabetArray(2))
Swap(A, B) → (B,A)

alphabetArray(1)=B
alphabetArray(2)=A

Thus, the values held in each array position were switched with each other. If this same process is performed over and over again using a psuedo random routine, it will become difficult to tell what value is held in any of the array positions.

(c)KSA

The Key Scheduling Algorithm is the first part of the encryption process. The following is the algorithm actually used in RC4, followed by a detailed annotation.

(d)Algorithm

```
1. Assume N = 256
2. K[] = Secrete Key array
3. Initialization:
4. For i = 0  to N – 1
5.      S[i] = i
6. j = 0
7. Scrambling:
8. For i = 0 ... N – 1
9.     j = j + S[i] + K[i]
10.    Swap(S[i], S[j])
```

(d)Explanation

1. N is an index value. It determines how strong the scrambling process is. WEP uses a value of 256.

2. K is the letter used to symbolize the secret key array. In the case of a five-character, pre-shared key, this value would be the three-character IV + five-character pre-shared key → eight-character secret key. Each character is held in the corresponding K position. This value does not get scrambled.

3. This starts the initialization of the KSA. It is used to seed the empty State (S[]) array with values 0[nd]255.

4. This is the start of the loop process that increases the value of i each time the algorithm loops.

5. Once it is done, the S array will hold values 0[nd]255 in corresponding array position 0[nd]255.

6. j is used to hold a value during the scrambling process, but it must first be initialized to ensure that it always starts at 0.

7. This starts the scrambling process that creates the psuedo random S array from the previously seeded S array.

8. Another loop that ensures the scrambling process occurs 256 times.

9. This is the equation used to merge the properties of the secret key with the state array (S[]) to create a psuedo random number, which is assigned to j.

10. Finally, a swap function is performed to swap the value held in S[i] with the value held in S[j].

Thus, it is not an overly complex process. Subjecting the secret key to simple math generates the psuedo random state array. The next part takes this array and creates a stream of data that is used to encrypt the data to be sent over the airwaves.

PRGA

The PRGA (Psuedo Random Generation Algorithm) is the part of the RC4 process that outputs a streaming key based on the KSA's psuedo random state array. This streaming key is then merged with the plaintext data to create a stream of data that is encrypted. Following is the algorithm and its explanation:

(d)Algorithm

```
1. Initialization:
2. i = 0
3. j = 0
4. Generation Loop:
5.     i = i + 1
6.     j = j + S[i]
7.     Swap(S[i], S[j])
```

<u>8. Output z = S[S[i] + S[j]]</u>
<u>9. Output XORed with data</u>

(d)Explanation

1. Again, before using the PRGA, the i and j values must be initialized.

2. i initialized to 0.

3. j initialized to 0.

4. This starts the stream-generation processes. It will continue until there is no more data, which in WEP's case is the end of the packet of data[md]or about 1,500 bytes.

5. i is added to itself to keep a running value used in the swap process.

   Note: This value will ALWAYS equal 1 the first time through the PRGA loop (i = i + 1 → i = 0 + 1 = 1).

6. j is used to hold the psuedo random number in the S[] position, with the previous S[] added to it.

   Note: This value will ALWAYS hold the value held in S[1] for the first iteration of the PRGA (j = j + S[i] → j = 0 + S[1]).

7. Another swap function is performed that switches the values held in the i position and j position of the state array.

8. z is calculated based on an addition of the value held in the state array, as represented by the addition of the values held in S[i] added to S[j].

9. Finally, the z value is XORed with the plaintext to create a new and encrypted value. This can be represented by the equation encrypted data = z XOR plaintext.

   Note: XOR merely requires that you know ANY two of the values to deduce the third. In other words, if the plaintext is known and the encrypted data is captured by a sniffer, a hacker can deduce the z value outputted by the PRGA.

(c)CRC

There is one final part of the data-transmission process that needs to be mentioned due to the fact that it adds additional data to the packet. This is the CRC, or Cyclic Redundancy Checksum value.

When a packet is sent across a network, there has to be a way for the receiving party to know that the packet was not altered or corrupted in transmission. This is accomplished via the CRC. Before the data is packaged and sent the CRC fingerprints the data and appends this value to it. Once the packet is received, the CRC value is removed, and a NEW CRC value is calculated on the received data. If the NEW CRC value matches the ORIGINAL CRC value, the packet is assumed to be complete; otherwise, the packet is considered corrupted and is dumped. As you will see next, this does affect the encryption process.

Cracking WEP

To review:

[lb] The IV is sent as plaintext with the encrypted packet. Therefore, *anyone* can easily sniff this information out of the airwaves and thus learn the first three characters of the secret key.

[lb] Both the KSA and PRGA leak information during the first few iterations of their algorithm. The i will always be 1, and j will always equal S[1] for the first iteration of the PRGA, and the KSA is easily duplicable for the first three iterations due to the fact that the first three characters of the secret key are passed as plaintext.

[lb] XOR is a simple process that can be easily used to deduce any unknown value if the other two values are known.

In addition to these previously explained points, there are several more that make WEP dangerous:

[lb] There is a 5% probability that the values held in S[0][nd]S[3] will *not* change after the first three iterations of the KSA. In other words, any hacker can guess what will happen during the KSA process with a 5% likelihood of being correct.

[lb] The first value of the encrypted data is always the SNAP header, which equals “AA” in hex or “170” in decimal form. This means that by sniffing the first byte of encrypted text and XORing it with 170, any hacker can deduce the first output byte of the PRGA.

[lb] In the WEP encryption process, it has been determined that a certain format of an IV indicates that it is a weak IV and subject to cracking. The format is (B + 3, 255, x) where B is the byte of the secret key being cracked. However, we know the first three characters due to the IV, so we want to crack the pre-shared password that starts after the IV. The 255 value indicates that the KSA is at a vulnerable point in the algorithm, and the value “x” can be any value.

Now that these points have been provided, consider how a hacker would use this knowledge to crack WEP.

(c)Walking Through the KSA

As previously mentioned, the IV is sent as plaintext. This can be easily sniffed out of the air and used to re-create the first three iterations of the KSA. The following is a technical example:

Before getting into the details, consider the following definitions:

[lb] The captured weak IV is 3, 255, 7. This value was chosen for this example because it was tested and is known to be a true, weak IV.

[lb] The pre-shared password is 22222. Although a hacker would not know this before cracking WEP, it is important to understanding this example.

[lb] N is 256.

[lb] If a value exists that is greater than N (256), a modulus operation must be performed on it. This divides the number by 256, which results in a leftover number called the modulus. This is the value that is passed on through the calculation.

[lb] The initialization process of the state array has already occurred and has seeded the state array with the 256 values.

First, consider the key array as a hacker would see it after capturing the IV:

K[0]=3	K[1]=255	K[2]=7	K[3]=?	K[4]=?	K[5]=?	K[6]=?	K[7]=?

Next, we need to define and track the state array values, i value, and j value. This will be done before each loop is processed in order to demonstrate how the values change. We will not show all 256 state array values because they are useless to the WEP-cracking process. Only the first four state array values and any value that has changed will be shown.

KSA loop 1

i=0	j=0	S[0]=0	S[1]=1	S[2]=2	S[3]=3		

j=j + S[i] + K[i mod l] = 0 + S[0] + K[0] = 0 + 0 + 3 = 3 → j = 3

In this equation, you can see that the j and i value were 0, which is used by the S[] array (S[0] = 0) and the K[] array (K[0] = 3). This resulted in the values of 0, 0, and 3 being added together to assign the value of 3 to j. This value is then passed on to the swap function below.

i=0, j=3

Swap (S[i], S[j]) → Swap (S[0] , S[3]) → S[0] = 0 , S[3] = 3 → S[0] = 3 , S[3] = 0

In this process, note that the values held in S[0] and S[3] are swapped. This is an important process to watch, but remember there is a 5% chance that the values held in S[0] → S[3] will not change after the first 4 KSA/PRGA loops.

KSA loop 2

i=1	j=3	S[0]=3	S[1]=1	S[2]=2	S[3]=0		

j=j + S[i] + K[i mod l] = 3 + S[1] + K[1 mod 8] = 3 + 1 + 255 = 259 mod 256 = 3 → j = 3

i=1, j=3

Swap(S[i], S[j]) → Swap (S[1] , S[3]) → S[1]=1 , S[3]=0 → S[1]=0 , S[3]=1

Note that in this loop the value of i increases by one and that a modulus operation was performed to determine the value of j. It is only coincidental that j = 3 again.

KSA loop 3

i=2	j=3	S[0]=3	S[1]=0	S[2]=2	S[3]=1		

j=j + S[i] + K[i mod l] = 3 + S[2] + K[2] = 3 + 2 + 7 = 12 → j = 12

i=2, j=12

Swap(S[i], S[j]) → Swap (S[2] , S[12]) → S[2]=2 , S[12]=12 → S[2]=12 , S[12]=2
Note that up to this point, only KNOWN values are used. Any hacker can reproduce this process up to this point. However, in the next step, the secret key is unknown, so the hacker has to stop.

KSA loop 4

i=3	j=12	S[0]=3	S[1]=0	S[2]=12	S[3]=1	S[12]=2	

j=j + S[i] + K[i mod l] = 12 + S[3] + K[3] = 12 + 1 + ? = ?

i=3, j=?

Swap(S[i], S[j]) → Swap (S[3] , S[?]) → S[3]=1 , S[?]=? → S[3]=?? , S[??]=1

So, now the hacker is up against a wall. However, what if there was a way to determine the j value at this point? Fortunately for the hacker, there is a way. A simple XOR calculation will determine this value from the first iteration of the PRGA process.

Knowing this, consider the XOR process that creates the encrypted data. The final step of the RC4 process is to XOR a PRGA byte with a byte of the plaintext data. Since XOR works in both directions, we also know that we can deduce the first byte of the PRGA if we XOR the first byte of the encrypted data with the first byte of plaintext. Fortunately, for a hacker this is easy thanks to the SNAP header (170 in decimal) and the use of a sniffer to capture the encrypted byte. In our example, we will provide the captured encrypted byte value (165 in decimal), which changes from packet to packet. The following equation illustrates the XOR process:

z = 0xAA(SNAP) XOR Ciphertext byte1 = 170 (Dec) XOR 165 (Dec) = 15 ➔ z = 15

As a result of this XOR calculation, the hacker can deduce that the PRGA value is 15 (decimal). Now, he can reverse-engineer the PRGA process and use this to determine the missing j value. Recall the known loop values as they would occur entering loop 4 of the KSA. Remember that these values can be easily reproduced by the use of the IV values.

KSA loop 4

i=3	j=12	S[0]=3	S[1]=0	S[2]=12	S[3]=1	S[12]=2	

1. Initialization:
2. i=0
3. j=0
4. Generation:
5. i = i + 1 = 0 + 1 = 1
6. j = j + S[i] = 0 + S[1] = 0 + 0 = 0
7. Swap (S[i], S[j]) → Swap (S[1] , S[0]) → S[1]=0 , S[0]=3 → S[1]=3 , S[0]=0
8. z = S[S[i] + S[j]] = S[S[1] + S[0]] = S[3 + 0] = S[3] = ?
9. ?=15 ➔ S[3] =15 at KSA4

From the previous discussion, note that i will always equal 1 for the first iteration of the PRGA (line 5). This then means that j will always equal S[0] (line 6). As we can see from the KSA loop 4 input values, S[1] = 0. This then results in j being assigned the value of 0 (line 6). The values held in S[i] and S[j] are then swapped, which means that S[1] is swapped with S[0] resulting in S[1] = 3 and S[0] = 0 (line 7). These values are then added together and used to pull a value from the state array. In this case, the combined S[i] and S[j] values = 3 (line 8). However, the S[3] value referenced here is from the completion of KSA loop 4, which is unknown to us. Fortunately, due to the XOR process, we know that the resulting value is 15, which means that S[3] will equal 15 at the output of KSA loop 4. Knowing this, a hacker only needs to reverse the KSA loop 4 process to deduce the secret key value.

Let us now walk though this as a hacker would.

KSA loop 4

i=3	j=12	S[0]=3	S[1]=0	S[2]=12	S[3]=1	S[12]=2	

S[3]=15 , S[15]=S[3]t-1 → S[3]=15 , S[15]=1

First, we know that the final step in the KSA loop is to swap values. Knowing the values of the state array after loop 4 completes and before it starts is important. Thanks to the XOR weakness, we know S[3] will equal 15, and we can make an educated guess that S[15] will hold the value held by S[3] before loop 4, which is 1 in this case. As a result, the hacker can deduce that S[3]=15 and S[15]=1 after the swap.

$\text{Swap}(S[3], S[15]) \rightarrow S[3]=1, S[15]=15$

Next, the hacker swaps the values held in these positions, which leaves S[3] equaling 1.

$j=j + S[i] + K[i \bmod 256] = 12 + S[3] + K[3] = 12 + 1 + K[3] = 15$

A hacker then plugs the values into the equation that would produce the j value. This fills in all the fields except the value of the secret key array.

$\rightarrow K[3] = 15 - 12 - 1 = 2$

After a simple reverse calculation, the value 2 is produced, which is the first byte of our secret key!

Simplifying the Process

This was a down-and-dirty look at how a hacker could deduce a secret key, byte by byte. However, WEPCrack (which was written as an educational tool) and AirSnort are both programs that can crack the secret key in a matter of seconds if enough data is present. The trick lies in the data.

Due to the requirements, roughly 7GB of data must be captured, on average, to crack the password. This is quite a bit of information. In fact, most home users and small businesses will have a tough time meeting this mark in two weeks. However, if a WLAN were operating at maximum flux, it can send this much data in two to four hours.

As this example illustrates, WEP can easily be cracked. Thus, if implementing .NET Server (or XP) with wireless functionality, never rely solely on WEP to protect your WLAN traffic. Instead, consider a more complete solution including VPN and RADIUS (which are beyond the scope of this paper).

Policy Changes

Death of the Microsoft Security Partners Program (MSPP)

The Microsoft Security Partners Program was designed to recruit elite security groups of all sizes to the Windows banner. Unfortunately, the MSPP ended before it ever got started. In October 2001, Microsoft abruptly ended the MSPP for unclear reasons.

On Nov 8, 2001, Microsoft announced that its aborted MSPP would be rolled into its existing, well-established Microsoft Gold Certified Partner Program. To this end, Microsoft formed the “Microsoft Gold Certified Partner for Security Solutions.” Category. Unfortunately, with this move Microsoft effectively barred entry to smaller security consulting groups that do not have the money or the manpower to be a member.

This is unfortunate, because smaller, independent security groups do a good deal of the best security testing.

Changes to the Microsoft Gold Certified Partner for Security Solutions (CPSS) from the MSPP included the following:

- Requires at least four (4) MCSE or MCSD certified technicians on staff (increased from two (2) in the MSPP).
- Requires an annual fee of approximately $1,450 - $2,300, which varies from region to region around the world (currently $1,450 per year in the USA).
- Requires more client references; also requires informing Microsoft about specific security customer details, including project description and work performed.
- Requires partners to sign a new full disclosure "gag rule" (described below), which prevents partners from publicly disclosing vulnerabilities found in Microsoft software.

Full Disclosure GAG Rule

On November 9, 2001, Microsoft surprised the security world by announcing a Coalition with five leading security companies. These firms include Bindview, Foundstone, Guardent, @Stake, and Internet Security Systems. To join the Coalition a member firm must sign a "gag rule" that waives their right to full disclosure of security vulnerabilities that have hurt Microsoft's reputation so severely.

Microsoft announced that it was leveraging the new coalition to create an Internet standard "gag rule" to prevent security experts from publicly revealing such vulnerabilities in Microsoft software. Coincident with the .NET initiative, Microsoft announced that it would formally oppose the "full disclosure" movement through various means, including spearheading the creation of new industry standards (RFCs) for non-disclosure.

Unintentional Anti-Competitive Pressure

Already steeped in controversy, the young security Coalition took a more menacing turn when one of its members damaged an open-source competitor of Microsoft. On June 17, 2002, Coalition member Internet Security Systems (ISS) posted a severe Apache vulnerability to the BugTraq mailing list, with barely any advance notice to Apache.org. It is a dark incident that no Apache administrator (especially those 32-bit users who stayed at work past midnight trying to apply a binary patch that did not work) is soon likely to forget.

How can Microsoft possibly be to blame for this colossal ISS *faux pas*? Legally, they are not. Ethically, however, the incident raises serious questions. For example, ISS likely derives economic benefit from its association with Microsoft's Coalition (at the very least in the form of publicity), and in return ISS helps protect Microsoft security from damaging public embarrassment. Here is a paradox: with one hand, ISS shields Microsoft with the gag rule, while with the other hand, ISS strikes one of Microsoft's greatest competitors in violation of that same gag rule. This paradox, when viewed in the light of the economic benefits ISS receives from Microsoft, could raise ethical concerns.

Summary

In summary, Windows .NET Server has the potential to be a secure operating system. In this paper we examined potential attacks against the architecture. In fact, Microsoft itself keeps delaying the release of .NET Server, citing security reasons. However, it is encouraging to note that no major vulnerabilities have yet been found in the beta tests of .NET Server. Thus, Microsoft should consider moving confidently towards final public release. Invariably, critical security flaws will emerge, but such flaws are universal, and they can be fixed.

More so than any fleeting vulnerability, it is the administrator's perception of the policies surrounding .NET Server that may be more damaging to the security image of the OS. For example, when someone reports an exploit, the best policy for Microsoft might be to quickly an honestly admit the error, to give full and public credit, and to work diligently and openly for a solution. Johannes Westerink, for example, reported a .NET cross-site scripting and full path disclosure vulnerability to the Microsoft Security Response Center in the summer of 2001. After waiting six months for a response that never came, Westerink in frustration posted the vulnerability publicly.

Accepting such an "open" ethic for vulnerability handling can be more advantageous that trying to influence external conditions, such as attempting to force industry standards for full disclosure. Similarly, pressuring critics with economic disadvantage (i.e., the "Gag Rule" requirement to be in the security Coalition), or tempting allies with the prospect of financial gain (i.e., the economic benefits of the Coalition), might ultimately be counterproductive.

Finally, it is critical for an operating system to protect the user's privacy. Most importantly, vigilant protection of the user's individual rights will help ensure that .NET Server is widely adopted. Maintaining the integrity of such individual rights requires not only a full disclosure of license tracking algorithms, but also the immediate abandonment of membership in the BSA – an organization that will hopefully soon be exposed to the full daylight of public scrutiny

Authors

Dr. Cyrus Peikari is Chief Technology Officer of VirusMD Corporation. Seth Fogie is Director of Engineering at of VirusMD Corporation. Peikari and Fogie co-authored the first book ever written on .NET Server: Windows .NET Server Security Handbook from Prentice Hall PTR (ISBN 0130477265).

References

The Business Software Alliance. www.bsa.org.

"U.K. accountants take issue with BSA tactics." By Laura Rohde. IDG News Service\London Bureau. August 27, 1999.

"Say NO to the BSA." CBS Affiliate KRLD Radio, Dallas, TX, 2000.

"Waking the Sleeping Giant: Is Windows .NET Server Secure?" By Cyrus Peikari. *Secure Computing Magazine* (print edition). June 2002.

Windows .NET Server Security Handbook. By Cyrus Peikari and Seth Fogie. Prentice Hall, 2002.

Windows Internet Security: Protecting Your Critical Data. By Seth Fogie and Cyrus Peikari. Prentice Hall, 2002.

"Erroneous VeriSign-Issued Digital Certificates Pose Spoofing Hazard." Microsoft Security Bulletin MS01-017. March 28, 2001.

"WPA Activation Horror" excerpt reprinted with permission from Robert Robinson. Newsgroup: microsoft.public.whistler.advanced-server.general. April 27, 2002.

"Is Windows XP's 'Product Activation' A Privacy Risk?" by Fred Langa. *Information Week*, Aug. 20, 2001.

"Feasibility of attacking Windows 2000 Kerberos Passwords." Excerpt reprinted with permision from Frank O'Dwyer. March 5, 2002.

"Is .NET Server Really 'Trustworthy'?" By Zubair Alexander. *InformIT.com.* May 01, 2002.

"Tamperproofing of Chip Card." Ross J. Anderson, Cambridge University Computer Laboratory.

"Pirates Cash In on Weak Chips." by James Glave. *Wired News*. May 22, 1998

"Tamper Resistance - a Cautionary Note." By Ross Anderson, Markus Kuhn. Cambridge University Computer Laboratory

PKI - Breaking the Yellow Lock” by Richard Forno. *SecurityFocus*, Feb 13 2002.

“50 Ways to Defeat PKI” by Fred Cohen. www.all.net

“Apache Admins Screwed by Premature Vulnerability Report” by Thomas C Greene. *The Register.* June 18, 2002.

Note: