



**SwarmSecure**  
Serious Security for Serious Business

## **Swarm Secure Whitepaper:**

### **Basic Forensic Analysis with Unix Tools You ALREADY Use Every Day**

A step-by-step guide to basic analysis of a potentially compromised system using open source utilities that are typically supplied with all Unix distributions.

Gary Golomb  
Managing Partner  
[gary@swarmsecure.com](mailto:gary@swarmsecure.com)

<http://www.swarmsecure.com/>

It's Monday morning. After going through the 78 pieces of spam and 396 messages from the Unix and security lists that arrived over the weekend, you check your IDS console. Hrmmm... An event not related to the worm-of-the-month. That's odd. In looking further at the captured packets, you see a sensitive configuration file was sent back to the potential attacker as part of the same session. Uh oh... Time to Panic?

No. It's time to check the system in a level-headed manor. The hint may have come from an IDS console, erratic system behavior, or a suspicious log entry, but when the time comes, staying clam and not jumping to conclusions is step number one! In the following sections, you'll be presented with methods for performing a basic analysis of a potentially compromised system. While you can glean a huge amount of information using standardly supplied open source tools, take heed; this does not replace a solid incidence response plan developed with the support of your company's (or consultant's) legal council!

## Disk Imaging First?

To image, or not to image. That is the argument.

It is commonly recommended to yank the power cord from your system if you suspect it has been compromised. There are several reasons for this. When the power is suddenly cut from the system, the files on the system will be preserved on the hard drive in their exact state at the time of discovery. When shutting down a system, not only could the shutdown process destroy evidence, but there could also be "trapdoors" left by the attacker to be executed in the event the system is shutdown.

However, in production environments, immediately pulling the power is frequently not possible. Additionally, your goal might not be to preserve "evidence" for attacker prosecution. Evidence is also useful for discovery, remediation, and training of system administrators and other security personal. Also, in some configurations, it might be necessary to shutdown before drives write the data stored in their cache to the actual disks. Ensure you discuss legal issues such as this with your local council. This is VERY important since it will directly impact how you carry out your investigation. So, is the next step imaging the hard drive to somewhere else for off-line analysis?

If you have the space to mirror drives every time you suspect something suspicious is happening, then by all means do so! However, if you are like the rest of us, you will need to do some cursory analysis first.

## Looking for Clues - Online Analysis

First and foremost - do NOT to trust anything on the suspect system. If it has been compromised, chances are high that all the tools you'll see used in this section have been replaced with trojans. The attacker-supplied binaries will hide exactly the types of activity you'll be looking for. It is imperative to use your own statically compiled and trusted binaries for the examination - including the shell (sh, bash, etc). You can either obtain the source to all the tools described herein and statically compile them, or take them from existing forensic toolkits. For a how-to on statically compiling binaries in Linux, see [http://www.incident-response.org/howto\\_1.htm](http://www.incident-response.org/howto_1.htm) and for Solaris, see <http://www.deer-run.com/~hal/sol-static.txt>. Precompiled binaries are also available at <http://www.incident-response.org/>.

Once you have created a CD with your tools, you can mount it as normal. Start the statically compiled shell you copied to the CD before proceeding any further. For our examples, we will be using bash. The potentially compromised host is a Red Hat 7.3 system. You will need to set your PATH to the trusted binaries and libraries (if used) on CD also.

```
# mount /dev/cdrom/  
# /mnt/cdrom/lin2-4_x86/bin/bash  
# PATH="/mnt/cdrom/lin2-4_x86/bin:$PATH"  
# LDLIBRARYPATH="/mnt/cdrom/lin2-4_x86/lib:$LDLIBRARYPATH"  
# export PATH  
# export LDLIBRARYPATH
```

Example 1: mounting CD with trusted binaries and setting shell path to those binaries and libraries.

Since you're still on the live and potentially compromised system, it might be a good idea to keep a low profile while conducting your investigation. If the attacker is still on the system, it would be wise to not alert them to your

activity. One method for doing this is to issue commands while inside the vi editor. If the attacker is watching the processes you (the admin) are executing, they should only see vi. Once inside vi, type " :!*command*" to use this method.

```
# vi working
:! ls -al
total 0
-rw-r--r-- 1 root  root   0 Jan 29 18:36  somefile
Hit ENTER or type command to continue
```

Example 2: Executing commands inside of vi.

In addition to normal places such as messages and secure logs, a good place to start your investigation is with the `/etc/passwd` and `/etc/shadow` files. Look in these files for deleted passwords or users that have a UID/GID of zero. On the system we are examining, the following entry was found in `/etc/passwd`.

```
resource:x:0:0:::/dev/.resource:/bin/bash
```

This alone is enough reason to image the drive for a more in-depth offline analysis, however for the purpose of this article we will poke around a little further. The next step would be to look for rouge processes. Use both `ps` and `netstat`. Since the attacker will most likely want to return to the system, chances are high you can find a daemon of some kind running. This is where the importance of your trusted binaries come-in. `ps`, `netstat`, and related programs will most certainly be trojaned to hide the presence of such daemons if they exist.

```
# cd /mnt/cdrom/lin2-4_x86/bin/      <---- trusted binaries
# ./netstat -an | ./grep -i tcp
tcp    0    0    0.0.0.0:22    0.0.0.0:*    LISTEN
tcp    0    0    0.0.0.0:80    0.0.0.0:*    LISTEN
tcp    0    0    0.0.0.0:443   0.0.0.0:*    LISTEN
tcp    0    0    0.0.0.0:2006   0.0.0.0:*    LISTEN

# ./netstat -an | ./grep -i udp
udp    0    0    0.0.0.0:624   0.0.0.0:*
```

Example 3: Open TCP and UDP ports.

Since TCP 2006 is not a familiar port, we can use lsof to see what is listening on that port.

```
# ./lsof | ./grep 2006
sshd    839    root    3u    IPv4    1309    TCP *:2006 (LISTEN)
```

Example 4: Using lsof to determine what service is listening on a certain port.

Finding a SSH daemon listening on a high port is almost certainly a sign of a compromised system. Attackers will use some kind of backdoor that gives them local shell access to a system. If that backdoor also encrypts their activity from network IDSs, then all the better. SSH is a perfect fit for such activity!

This is a safe point to make the decision to image the drive and begin a more in-depth analysis. Before continuing, we are going to perform one more trick. Since we appear to have a rogue SSH daemon process (pid: 839) running on a high port, let's copy the running sshd binary (using the techniques described in the next section) by pulling it out of memory to perform file analysis on later. This is done by copying the "exe" file from the directory /proc/<pid>. On Solaris, object/a.out is the executable file.

```
# cd /proc/839/
# ls
cmdline  cwd  environ  exe  fd  maps  mem  mounts  stat  statm  status
          ^
          |----- Copy this file
```

Example 5: Pulling the binary of a running program from memory.

## Moving and Mounting Images

In this section, we will back up the drives of the compromised system to a separate system to perform a more in-depth analysis of the system. However, before moving partition images, you need to know how the disk is partitioned. For this task use `fdisk`. It is important to use the `-l` flag with `fdisk` to prevent the utility from mounting the disk. `'df'` can be used in some cases to help you match devices with labels.

```
# fdisk -l
Disk /dev/hda: 255 heads, 63 sectors, 7296 cylinders
Units = cylinders of 16065 * 512 bytes

   Device Boot      Start   End  Blocks  Id System
/dev/hda1    *          1   1275   10241406    7 HPFS/NTFS
/dev/hda2             1276   1288     104422+   83  Linux
/dev/hda3             1289   1543     2048287+   82  Linux swap
/dev/hda4             1544   7296     46210972+    f  Win95 Ext'd
(LBA)
/dev/hda5             1544   7296     46210941   83  Linux
```

Example 6: Using `fdisk` to discover the partition scheme of the hard disk.

Before copying images, you need to create md5 hashes of the partitions. This is to ensure that once the copy process is completed, none of the potential evidence was modified during the copy process. Copy the results of the hash to your analysis system using the same methods used to copy the images. This is done with `netcat` (for added security, `cryptcat` can be used but is not included by default with most distributions). First, you'll need to setup a `netcat` listener on your analysis system. Here is how the listener will receive the results of the md5 hash of `/dev/hda1`.

```
# nc -l -p 5555 > md5_before_img.hda1
```

On the compromised system:

```
# ./md5sum /dev/hda1 | ./nc 192.168.0.10 5555
```

Repeat this process for `hda1` through `hda5` (or for the number and types of partitions on your system). Now we're ready to image the partitions. For this task we'll replace `md5sum` with `dd`. On the analysis system:

```
# nc -l -p 5555 > hda1.image
```

And on the compromised system:

```
# ./dd if=/dev/hda1 | ./nc 192.168.0.10 5555
```

Example 7: Using netcat to transfer data across a network.

Again, repeat for each partition. Once you have copied the images to the analysis system, you need to log the results of md5sum on the image files themselves, for auditing purposes. This time, md5sum the *hda#.image* files you have created and compare to the original results. Everything should be the same.

Once the integrity of the images have been verified, make a copy of each image. Perform ALL of your analysis work on these copies, and NOT the originally created images.

The last step in this process is to mount the newly acquired images for examination. If you are not familiar with these mount options, read the mount man page since they are important to understand. On the analysis system:

```
# mkdir ./mount/  
# mkdir ./mount/boot  
# mount -ro,loop,nodev,noexec,noatime,nosuid ./hda2.image  
./mount/boot  
# mkdir ./mount/root  
# mount -ro,loop,nodev,noexec,noatime,nosuid ./hda5.image  
./mount/root
```

Example 8: Mounting the disk images transferred over netcat on the analysis server.

Again, repeat this process for each partition to be analyzed on the examiner's system.

## Looking for Clues - Offline Analysis

At this point, 'find' will become your best friend. First, you need to account for certain types of files on the system. We'll start with all the hidden directories on the file system. It is also useful to include the timestamps on the files for correlation purposes.

```
# find /home/examine/mount/<label> -name ".*" -type d -
printf "%Tc %k %h/%f\n"

Wed 29 Jan 2003 01:37:15 PM EST 4 /dev.resource <----- BAD!
Wed 29 Jan 2003 01:30:03 PM EST 4 /dev/.resource.kde
Sat 25 Jan 2003 12:47:51 PM EST 4 /var/crash.ssh
Sat 25 Jan 2003 06:44:10 AM EST 4 /etc/skel.kde
Sat 25 Jan 2003 06:39:17 AM EST 4 /root.kde
Wed 29 Jan 2003 01:31:41 PM EST 4 /usr/bin.zeen
Wed 29 Jan 2003 01:31:42 PM EST 4 /usr/bin/.zeen.. <--BAD!
```

Example 7: finding hidden directories.

Many rootkits and attackers will hide data and files in /dev. Although /dev contains 1000's of files (making it a great hiding place), most of them are special block or character device files. Many rootkits can be exposed simply by looking for files that do not match the profile of files normally found in /dev. The example below shows this in addition to creating a time-line showing the order in which each file in the list of results was modified. The timestamp output is displayed as seconds since Jan 1, 1970, making it possible to sort by time.

```
# find /home/examine/mount/<label> -not -type b \
> -not -type c -printf "%T@ %k %h/%f\n" | sort -r

1043871279 88 /dev/
1043871279 0 /dev/gpmctl
1043870574 4 /dev/.resource/.bash_history
1043870535 4 /dev/ssh_random_seed
1043870533 0 /dev/log
```

```

1043870506 0 /dev/shm
1043865784 0 /dev/initctl
1043865435 4 /dev/.resource <----- BAD!
1043865102 4 /dev/.resource/cb-r00tkit
1043865101 4 /dev/srd0
1043865014 1052 /dev/.resource/cb-r00tkit.tgz

```

Example 8: Finding non-character and non-block devices in /dev.

Finally, all SUID root files should be accounted for. In the results of this command, you should be looking for programs that would give unprivileged users too much power over the system. A SUID bash shell is one such example.

```

# find /home/examine/mount/<label> \( -perm -002000 -o \
> -perm -004000 \) -type f -ls

```

We have identified several files that are presumably the rootkit(s) uploaded by the attacker, but now it is time to get a better idea of what kind of activity has taken place on the system. The following command will show all executable files owned by root that have recently been modified. This is a great way of identifying trojaned system binaries like ps, top, ls, bash, etc. Again, we'll sort the output based on time showing the order of modification of these binaries. Depending on how heavily the system is used, you may need to adjust the line count parameter to tail. On unused honeypots, it could be as low as 20. On some production systems, it may need to be thousands higher.

```

# find /home/examine/mount/<label> -type f -user 0 -perm +111 -
printf "%T@ %k %h/%f\n" | sort | tail -1500

1008275798 4 /usr/share/texmf/context/perlTk/mptopdf.pl
1008344861 36 /usr/bin/units
1008532089 8 /usr/bin/.zeen/.. /d <----- Files to focus on later
1008532096 8 /usr/bin/.zeen/.. /sl2 <--- |
1008532101 12 /usr/bin/.zeen/.. /statdx <--- |
1008532104 8 /usr/bin/.zeen/.. /v <--- |
1008532107 8 /usr/bin/.zeen/.. /write <--- |
1008532111 8 /usr/bin/.zeen/.. /wscan <--- |
1008532115 8 /usr/bin/.zeen/.. /wted <--- |
1008532118 32 /usr/bin/.zeen/.. /wu <--- |
1008532131 8 /dev/.resource/cb-r00tkit/fix <--- |
1008532572 24 /usr/bin/.zeen/.. /scan/strobe/strobe <--- |
1008534111 16 /dev/.resource/cb-r00tkit/encrypt <--- |
1008534111 32 /usr/bin/md5sum <----- Trojaned Binaries
1008534111 88 /usr/sbin/lsof <----- Trojaned Binaries

```

```

1008535262 40 /lib/libproc.so.2.0.6
1008537314 4 /bin/login <----- Trojaned Binaries
1008537613 4 /dev/.resource/cb-r00tkit/startfile <--|
1008537626 4 /dev/.resource/cb-r00tkit/lg <--|
1008610518 8 /usr/share/texmf/context/perlTk/fdf2tan.pl
1008611058 28 /usr/share/texmf/context/perlTk/texfont.pl
1008643721 4 /usr/share/doc/mrproject-0.5.1/NEWS
1008663591 4 /usr/share/doc/pygtk2-
1.99.8/examples/gobject/properties.py
1008695230 4 /usr/share/doc/pygtk2-
1.99.8/examples/gobject/signal.py
1009061415 4 /usr/bin/.zeen/.. /scan/1 <--|
1009725492 4 /dev/.resource/cb-r00tkit/remove <--|
1009940246 4 /usr/bin/pgpwrap
1009940246 4 /usr/bin/url_handler.sh
1009940246 8 /usr/bin/muttbug
1009940247 16 /usr/bin/urlview
1009940247 24 /usr/bin/pgpring

```

Example 9: Finding all created and modified executables owned by root.

After using the above command to identify a period of suspicious activity, you can reissue a more general search by modifying or dropping the "*-type f -user 0 -perm +111*" parameters. While it will return 1000's more results, you'll quickly be able to jump to the section you are interested in by using the timestamps from above. By doing this, you can see ALL files changed by the attacker and their activities. This can also show the types of configuration changes made to the system. Another interesting twist on the above example is using find to create a time-line of command/binary usage. Since `.bash_history` was probably erased, you can use find to create a time-line based on access timestamps. This in effect gives you a history replay.

```

# find /home/examine/mount/<label> -type f -perm +111 \
> -printf "%A@ %h/%f\n" | sort

```

However, one of the single most powerful techniques for finding trojaned binaries is not by looking at timestamps (as they can easily be modified to throw-off your investigation). An effective method is to examine the inodes of the files directly. This works well in directories like `/bin`, `/usr`, `/sbin`, and related directories. If you find a clump of binaries that look severely out of place, you can be sure those files require further investigation. For this, use `'ls'` in the directory in question.

```

# cd /home/examine/mount/root/bin
# ls -itl | sort | less

```

```

1321158 -rwxr-xr-x 1 root root 3744 Dec 16 2001 login
1321159 -rwxr-xr-x 1 root root 155462 Mar 24 2002 ls
1321163 -rwxr-xr-x 1 root root 54152 Apr 12 2002 netstat
1321165 -rwxr-xr-x 1 root root 62920 Apr 15 2002 ps
732969 -rwsr-xr-x 1 root root 35192 Apr 18 2002 ping
732970 -rwxr-xr-x 1 root mail 66492 Jun 24 2001 mail
732971 -rwxr-xr-x 1 root root 4236 Mar 31 2002 mktemp
732972 -rwxr-xr-x 1 root root 12952 Feb 26 2002 mt
732973 lrwxrwxrwx 1 root root 8 Jan 25 06:14
dnsdomainname -> hostname
732974 lrwxrwxrwx 1 root root 8 Jan 25 06:14
domainname -> hostname
732975 -rwxr-xr-x 1 root root 12426 Apr 12 2002
hostname
732977 lrwxrwxrwx 1 root root 8 Jan 25 06:14
nisdomainname -> hostname
732978 lrwxrwxrwx 1 root root 8 Jan 25 06:14
ypdomainname -> hostname
732979 -rwxr-xr-x 1 root root 16700 Sep 17 2001
setserial

```

Example 10: Examining inodes in a directory to find out-of-place binaries.

Looking at the above example, login, ls, netstat, and ps have inode entries (far left column) that are grossly out of place relative to the other files in this directory. As it so happens, these are just a few of the files that are commonly trojaned by rootkits. From this output, we can make an educated guess that these files have been trojaned or backdoored.

## File Analysis

Now that you've identified files modified by the attacker and her rootkit, it's time to paint a complete picture of what was done to the system and what the system was potentially going to be used for. Many of the modified and added files seen so far are straightforward, but not all of them.

Many of the newfound files are executable, but just executing them is the last way to go about discovering what the file does! While executing the file may become necessary for more advanced analysis and debugging (in a highly controlled environment), there are several tools at our disposal to get a cursory

idea of what we have on our hands. One example would be the file "1" found in the directory /usr/bin/.zeen/..\ /scan/ in example 9. Using the 'file' utility:

```
# file /usr/bin/.zeen/..\ /scan/1
/usr/bin/.zeen/..\ /scan/1: ELF 32-bit LSB executable, Intel
80386, version 1 (SYSV), statically linked, stripped
```

From this we see it's a statically compiled executable, but we still don't know what it does. Before looking further into "1," look at some of the other files in the same location:

```
# file /usr/bin/.zeen/..\ /cl
/usr/bin/.zeen/..\ /cl: Bourne-Again shell script text
executable
```

```
# file /usr/bin/.zeen/..\ /read
/usr/bin/.zeen/..\ /read: perl script text executable
```

```
# file /usr/bin/.zeen/..\ /scan/wus
/usr/bin/.zeen/..\ /scan/wus: ELF 32-bit LSB executable, Intel
80386, version 1 (SYSV), dynamically linked (uses shared libs),
not stripped
```

Example 11: Using the file command to identify files in a rootkit.

Since "cl" and "read" are text files, just open them in a text editor and read them! For the other files identified as interesting, you'll need to use 'strings' or a hex editor to start getting a clearer picture. Running strings on a typical binary will return quite a bit of output, but a patient eye will generally payoff. Among other things, here are a few of the interesting parts of "1" returned by strings:

```
# strings ./1

7350
: CWD %s
| CWD %s
CWD %s%c
RNFR: %d x 0x%08x (%d)
RNFR
350
building chunk: ([0x%08lx] = 0x%08lx) in %d bytes
220-
220
USER %s
331
```

```
PASS %s
230-
CWD %s
550
CWD ~/{.....}
250
CWD .
~//{.....}
bridge_dist = 0x%08lx
padchunk_size = 0x%08lx
fakechunk_size = 0x%08lx
padchunk_size = 0x%08lx
```

Example 12: Examining output of strings.

Hrmm.... Perhaps a Team Teso FTP exploit? (Notice the "7350" string at the top?) CWD, PASS, and USER are FTP commands. Continuing our search through the output:

[List truncated]

```
RedHat 6.1 (Cartman) [wu-ftp-2.5.0-9.rpm]
Version wu-2.6.0(1) Fri Jun 23 09:17:44 EDT 2000
RedHat 6.0|6.1|6.2 update [wu-ftp-2.6.0-14.6x.i386.rpm]
Version wu-2.6.0(1) Thu Oct 21 12:27:00 EDT 1999
RedHat 6.? [wu-ftp-2.6.0-1.i386.rpm]
Version wu-2.6.0(1) Fri Jun 23 09:22:33 EDT 2000
Mandrake 8.1 [wu-ftp-2.6.1-11mdk.i586.rpm]
Version wu-2.6.1(1) Wed Jan 10 07:07:00 CET 2001
Mandrake 7.2 update [wu-ftp-2.6.1-8.3mdk.i586.rpm]
Version wu-2.6.1(1) Mon Jan 15 20:52:49 CET 2001
Mandrake 6.0|6.1|7.0|7.1 update [wu-ftp-2.6.1-8.6mdk.i586.rpm]
Version wu-2.6.1(1) Mon Jan 29 08:04:31 PST 2001
Immunix 7.0 (Stolichnaya) [wu-ftp-2.6.1-6_imnx_2.rpm]
Version wu-2.6.0(1) Thu May 25 03:35:34 PDT 2000
Immunix 6.2 (Cartman) [wu-ftp-2.6.0-3_StackGuard.rpm]
Version wu-2.6.1(1) Sat Feb 24 01:43:53 GMT 2001
Debian sid [wu-ftp_2.6.1-5_i386.deb]
Version wu-2.6.0(1) Thu Feb 8 17:45:47 CET 2001
Debian potato [wu-ftp_2.6.0-5.3.deb]
Version wu-2.6.0(1) Fri Jun 23 08:07:11 CEST 2000
Caldera eDesktop|eServer|OpenLinux 2.3 update [wu-ftp-2.6.1-130L.i386.rpm]
```

Example 13: Examining output of strings.

Obviously this is a robust and friendly exploit. Finding output like this generally indicates it is a buffer overflow exploit. Buffer overflows can be difficult for

newer hackers to execute successfully since finding the required return addresses and offsets requires a familiarity with memory debugging. Finding a menu like the one above in an exploit usually means the return addresses and offsets for all the listed operating systems have been hard coded into the exploit, making it easier for novice hackers to exploit the vulnerability. Continuing our search, we finally find a usage statement:

```
usage: %s [-h] [-v] [-a] [-D] [-m]
        [-t <num>] [-u <user>] [-p <pass>] [-d host]
        [-L <retloc>] [-A <retaddr>]
-h      this help
-v      be verbose (default: off, twice for greater effect)
-a      AUTO mode (target from banner)
-D      DEBUG mode (waits for keypresses)
-m      enable mass mode (use with care)
-t num  choose target (0 for list, try -v or -v -v)
-u user  username to login to FTP (default: "ftp")
-p pass  password to use (default: "mozilla@")
-d dest  IP address or fqhn to connect to (default: 127.0.0.1)
-L loc   override target-supplied retloc (format: 0xdeadbeef)
-A addr  override target-supplied retaddr (format: 0xcafebabe)
7350wurm - x86/linux wuftp <= 2.6.1 remote root (version
0.2.2)
team teso (thx bnuts, tomas, synnergy.net !).
Compiled for MnM 01/12/2001..pr0t!
```

Example 13: Examining output of strings.

As we can see, it is indeed an ftp exploit by team teso! We can apply this methodology to most of the other files found on the system as well. Going through the other files would yield a script that automates scanning for FTP servers and exploiting them using this FTP exploit. In other words, this system was going to be used to launch a mass-exploitation of other systems on the Internet. These other system could eventually be used to build a huge Distributed Denial of Service network, or as a large number of leap-pad systems for the attacker to launch an attack though that makes trace-back to them virtually impossible.

## The End?...

In this article, only a small handful of tools were used - all of which are available by default on almost every unix distribution. From these common tools, it was possible to quickly identify the system as being compromised and perform enough of an analysis to being recovering the attacker's rootkit, trojaned files, and create a more detailed picture of what the attacker did to the system and why.

Shown here is a very small fraction of everything involved in a good forensic investigation. In addition to accounting properly for evidence suitable for submission into a legal trial, documentation of findings, and general incident management, you may be faced with rootkits that do not lend themselves so easily to analysis. Needing to deal with Loadable Kernel Modules, low-level encrypted binaries (such as burneye and shiva encryption), and recovering deleted files/data are commonplace.

Collaborating with a solution provider like Swarm Secure brings benefits far beyond the initial stages of security solution design, testing, implementation, maintenance, and training. Your relationship with Swarm Secure ensures you have skilled and technical support available to you at all stages of your business's lifecycle.

## About Swarm Secure

Swarm Secure, Inc. is a full service enterprise security consulting firm headquartered centrally between Washington, DC and Baltimore, MD. Swarm Secure is dedicated to professionals looking to obtain best of breed security solutions including: recommendations, resources, products, professional services, training, and temporary or permanent placement opportunities.

At Swarm Secure we work with you to determine the appropriate solution for your unique situation given constraints such as people, time, and cost while considering technical, legal, and threat concerns. Most importantly, we focus on solutions that work for you. We do not bring cookie cutter approaches. We find the products, people and services that match your needs.

We provide custom designed security solutions for a variety of clients who operate in high risk, high threat, and high profile environments. Through responsive, specialized teams of technology experts, Swarm Secure is able to provide a one-stop availability for a comprehensive set of security consulting services.

The partnership of engineers who now provide the framework for Swarm Secure have years of experience in Intrusion Detection research, signature development, exploit and compromised systems analysis, and enterprise security design/implementation. We cover all aspects of security solution analysis, design, product selection, solution deployment & management, testing, training, documentation, certifications, and post installation support services. Our team of highly analytical consultants are ready to work intimately with you on developing a personalized solution for your environment.