Background

The GT.M Group at Fidelity National Information Services (FIS) recently received a report of a GT.M security vulnerability. This advisory provides you with information about the vulnerability, actions we have taken to remedy the situation and actions you can take.

The protection of client data and security is foremost within the FIS organization and is key to our ongoing success. Therefore, we are committed to resolving this matter not only in a timely manner, but an effective one. We regret any inconvenience this may cause you. If you have any further questions or concerns, please do contact us.

Revision History

- October 27, 2008 – Initial publication of advisory.
- October 31, 2008 – Updated with details of remediation by wrapping.

Summary

`gtmsecshr` is a helper program included with GT.M that must be installed owned by root and with the setuid bit on. For example, one purpose is to enable a GT.M process to send a signal to a GT.M process with a different user id, to signal the release of an M lock by one process to a process awaiting that lock. If the two processes have the same user id, the signal is sent directly; if they are different user ids, UNIX/Linux does not permit the signal to be sent and `gtmsecshr` is invoked to send the signal on behalf of the process that seeks to send the signal.

We have identified GT.M code for accessing environment variables that does not protect itself against a buffer overflow. Setting an environment variable to an excessively long string causes an internal buffer overflow. As the source code for accessing environment variables is shared by `mumps`, `mupip` & `lke` processes, as well as `gtmsecshr`, the vulnerability applies to all of them. However, since the other processes do not normally operate as root, vulnerability with `gtmsecshr` is the concern.

Note that all prior UNIX/Linux GT.M releases, through V5.3-002/V5.3-002A are potentially vulnerable to the possibility of an exploit by someone who has programming (shell) access to a system. There is no vulnerability to a remote attack, and no vulnerability to a user who does not have shell access (i.e., the vulnerability is not open to server processes that are invoked via `inetd/xinetd`, processes that service messages received through a communications monitor such as MTM, or to users who immediately go into an application program after login, and are blocked from shell access when connected to the system).

Follow up

The buffer overflow will be fixed in GT.M V5.3-003, which will be released in December, 2008.

We have implemented a remediation for all prior GT.M versions that completely blocks any exploit of this vulnerability by "wrapping" `gtmsecshr` and preventing the buffer overflow. It works as follows:

- The `gtmsecshr` program in an existing GT.M installation is moved to a new directory called `gtmsecshrdir`. The `gtmsecshrdir` sub-directory is readable and executable only by root and the `gtmsecshr` program is made executable only by root.

- A small C wrapper program clears the environment of all variables except `$gtm_dist`, $`gtmdbglvl`, `$gtm_log` & `$gtm_tmp`. It sets `$gtm_dist` to the absolute path of `gtmsecshrdir/gtmsecshr`. After ensuring that these environment variables are within length limits, it then executes `gtmsecshrdir/gtmsecshr`. An executable image of this C program replaces the original `gtmsecshr` in the GT.M distribution directory, i.e., owned by root, with the name `gtmsecshr` and with the setuid bit on.

Since an environment variable cannot be used to overflow a buffer in `gtmsecshr` once the wrapper is in place, the vulnerability is blocked from being exploited. Specific instructions are provided below.

Other Suggested Actions

Although the wrapper blocks the vulnerability from exploits, and other remediation is not required, we generally recommend the additional measures below as being good practice, since good security is built on layers of protection.

1. Ensure that only authorized persons have access to the system, and otherwise secure your system per current industry best practices.

2. Ensure that shell access is available only to those who need it, and users who don't need shell access are taken immediately the application and blocked from shell access.

3. Restrict the ability to run GT.M to members of a group, if all users of the system do not need to run GT.M.

4. Implement / turn on address space layout randomization (ASLR – see http://en.wikipedia.org/wiki/Address_space_layout_randomization). Linux kernels since 2.6.12 provide a form of address space layout

randomization, and other layered software provides better randomization. Layered software can also provide ASLR for older Linux kernels. For UNIX systems, check the documentation for ASLR options that may be available to you.

5. On Linux, disable the default execution of code in stack and heap space (there are a number of ways of implementing this – consult your operating system documentation). Note that on Linux GT.M `mumps` processes through V5.3-000 required the ability to execute code in heap space. Even with a default setting to disable the execution of code in heap space, `mumps` processes can still be permitted to execute code in heap space, for example, with a command such as `execstack -s $gtm_dist/libgtmshr.so` (the command will vary depending on the mechanism used to implement the restriction). Effective GT.M V5.3-001, `mumps` processes on Linux no longer need the ability to execute code in heap space. Disabling execution of code in stack / heap space effectively blocks exploits that use buffer overflow vulnerabilities to execute rogue code.

6. Consider additional layers of access controls, such as those available with Access Control Lists (ACLs), Security Enhanced Linux (SELinux), Trusted Solaris, Trusted AIX, etc. Consult your operating system documentation.

Implementing the gtmsecshr wrapper

The wrapper is distributed both as source code and as an executable image. The executable images should run as-is on common versions of UNIX/Linux operating systems for which GT.M is available. However, given the range of operating system versions in use, of varying ages, it is possible that the executable images may not run on your system. If that is the case, you will need a C compiler to create your own executable image on the operating system version you use. The source code is ANSI standard C and the same source code has been built and tested on a range of operating system versions and platforms.

Obtain the wrapper

- FIS Profile licensees with active support agreements: obtain the wrapper from your normal Profile support channel. The wrapper is considered part of GT.M and licensed to you under your current Profile/GT.M license.

- GT.M licensees with active support agreements: download the wrapper from the GT.M FTP site (contact GT.M Support for the current access information if you don't have it). Verify the checksum for the downloaded archive with the standard UNIX/Linux `cksum` program (use the native `cksum` program for each operating system):
```
3419770628 4992 gtmsecshr_wrapper_01_linux_i686.tar.gz
425326214 5270 gtmsecshr_wrapper_01_linux_x8664.tar.gz
```

```
2851691682 24926 gtmsecshr_wrapper_01_aix_rs6000.tar.gz
2813816232 5904 gtmsecshr_wrapper_01_solaris_9.tar.gz
2062029970 6294 gtmsecshr_wrapper_01_solaris_10.tar.gz
1532578548 6167 gtmsecshr_wrapper_01_linux_ia64.tar.gz
2248448569 6831 gtmsecshr_wrapper_01_hpux_ia64.tar.gz
727882187 10582 gtmsecshr_wrapper_01_hpux_parisc.tar.gz
1182494145 6112 gtmsecshr_wrapper_01_osf1_alpha.tar.gz
```

The wrapper is considered part of GT.M and licensed to you under your current GT.M license.

- All other GT.M users, including those licensing GT.M under GPL v2, GPL v3 and AGPL v3 and without active support agreements: obtain the wrapper from the download section of the GT.M project at Source Forge (http://sourceforge.net/project/showfiles.php?group_id=11026&package_id=297369). Verify the checksum for the downloaded archive with the `cksum` program on your OS):
  ```
  208097648 17027 gtmsecshr_wrapper_01_linux_i686.tar.gz
  639131471 18228 gtmsecshr_wrapper_01_osf1_alpha.tar.gz
  ```
  The wrapper is licensed to you under AGPL v3 (provided in the `COPYING` file in the archive).

As a normal (non root) user, unpack the contents of the archive in a working directory. There will be two files, a source file `gtmsecshr_wrapper.c` and an executable file `gtmsechr` (if you obtained the wrapper from Source Forge, there will be a third file, `COPYING`).

(Optional) Compile the wrapper

In the event you need to, or want to, compile the wrapper to create your own executable image, the following are the commands we used. You are not required to use the same compilers that we used – any standard C compiler and linker/loader that can create an executable image should suffice.

Linux x86: `gcc -o gtmsecshr gtmsecshr_wrapper.c`

Linux x86_64: `gcc -o gtmsecshr gtmsecshr_wrapper.c`

AIX: `gcc -o gtmsecshr gtmsecshr_wrapper.c`

Solaris 9: `gcc -DPROVIDECLEARENV -DPROVIDESETENV -DPROVIDEUNSETENV -o gtmsecshr gtmsecshr_wrapper.c`

Solaris 10: `cc –DPROVIDECLEARENV -o gtmsecshr gtmsecshr_wrapper.c`

Linux Itanium: `gcc -o gtmsecshr gtmsecshr_wrapper.c`

HP-UX Itanium: `gcc -o gtmsecshr gtmsecshr_wrapper.c`

HP-UX HPPA: `cc –DPROVIDESETENV -o gtmsecshr gtmsecshr_wrapper.c`

Tru64: `gcc -o gtmsecshr gtmsecshr_wrapper.c`

Note: the C wrapper program uses functions `setenv()`, `unsetenv()` and `clearenv()`. These are typically provided by the operating system / C language subsystem. Implementations of those functions are provided within `gtmsecshr_wrapper.c`, but are not enabled by default because the functions provided by the operating system should be used preferentially. Where the OS does not provide a function, the one provided within `gtmsecshr_wrapper.c` can be enabled with a `-DPROVIDE`<routine-name> flag to the compiler. For example, we used `–DPROVIDECLEARENV` on Solaris 10 to enable the `putenv()` function within `gtmsecshr_wrapper.c`. Without it, the Solaris 10 linker would complain about not finding the symbol `clearenv`.

Install the wrapper

After bringing down all GT.M processes, perform the following steps as root. Adjust the commands according to the directories on your computer.

Change to the directory where GT.M is installed. You should see an existing setuid root `gtmsecshr` program.

```
root@PaMln2WL:~# cd /opt/lsb-gtm/V5.3-002_x86_64 ; ls -l gtmsecshr
-r-sr-xr-x 1 root bin 240048 2008-08-19 14:05 gtmsecshr
```

Create a directory `gtmsecshrdir` and set permissions to allow only root access to that directory and its contents.

```
root@PaMln2WL:/opt/lsb-gtm/V5.3-002_x86_64# mkdir gtmsecshrdir ; chmod g-rx,o-rx gtmsecshrdir
root@PaMln2WL:/opt/lsb-gtm/V5.3-002_x86_64# ls -ld gtmsecshrdir
drwx------ 2 root root 1 2008-10-30 18:28 gtmsecshrdir
```

Change the permissions of `gtmsecshr` and move it to `gtmsecshrdir`.

```
root@PaMln2WL:/opt/lsb-gtm/V5.3-002_x86_64# chmod g-rx,o-rx gtmsecshr ; mv gtmsecshr gtmsecshrdir/
```

```
root@PaMln2WL:/opt/lsb-gtm/V5.3-002_x86_64# ls -l gtmsecshrdir/gtmsecshr
-r-s------ 1 root bin 240048 2008-08-19 14:05 gtmsecshrdir/gtmsecshr
```

Copy the `gtmsecshr` program that you downloaded or that you compiled into the GT.M distribution directory and install it setuid root.

```
root@PaMln2WL:/opt/lsb-gtm/V5.3-002_x86_64# cp /tmp/tmp/gtmsecshr . ; chmod u+s gtmsecshr
root@PaMln2WL:/opt/lsb-gtm/V5.3-002_x86_64# ls -l gtmsecshr
-rwsr-xr-x 1 root root 10275 2008-10-30 18:38 gtmsecshr
```

Verify that the wrapper is working

As a normal (not root) user, verify that you can start up a `gtmseschr` process (one should not be running already).

```
kbhaskar@PaMln2WL:~$ ps -ef | grep gtmsecshr | grep -v grep
kbhaskar@PaMln2WL:~$ $gtm_dist/gtmsecshr
kbhaskar@PaMln2WL:~$ ps -ef | grep gtmsecshr | grep -v grep
root     16887     1  0 18:43 pts/1    00:00:00 gtmsecshr
```

The `gtmsecshr` process will shut itself down after one hour of inactivity.