GEORDI: A Handheld Tool For Remote System Administration

Stephen Okay – Road Knight Mobility Labs
Gale Pedowitz – Protura Consulting, Inc.

ABSTRACT

This paper discusses the design and implementation of a tool for allowing technical staff to perform diagnosis, triage and remediation of system problems from a commodity handheld device (e.g., a PalmOS PDA) with a wireless network connection using industry standard encryption and privilege management software. We argue that this model is equivalent to using a desktop or laptop from a security aspect but is more convenient and efficient given its minimal resource requirements, "instant-on" availability and usability from arbitrary locations.

We explore previous work in this area and posit that our solution offers significant advantages because it adopts the stylus and forms-based usage model prevalent on many PDAs rather than trying to overlay the classic command-line interface onto a system which was not designed to use it.

Finally, consideration is given to how small mobile systems, like the one used in this tool, will impact the task of system management in the future in terms of benefits and risks.

Introduction

The life of an on-call system administrator (sysadmin) is an interrupt-driven endeavor. When not beset by users or machines demanding his or her instant attention, he or she is surrounded by an array of pagers, PDAs, cell phones, and other messaging systems, all eager to alert him or her to fresh disasters. These seem to go off most frequently when the sysadmin is within range of being alerted about a problem but a significant distance from the nearest point from which to affect a solution.

No longer limited to displaying just the same numeric string much of the time, many of these devices are sophisticated two-way, programmable messaging systems with considerable memory and CPU power. Sometimes they aren't separate devices, but software built into cell phones or PDAs. Despite all these enhancements, the response to them is often the same as it has always been: to walk, run or drive to the nearest terminal to fix the problem. This is not only irritating to on-duty staff, it's wasteful and expensive, especially when the problem is often something as simple as "the $DAEMON died" or "$SERVER needs to be rebooted." In the current climate where users and equipment are scattered across time zones around the globe, the answer "I'll be there as soon as I can" is not an acceptable response.

The General External Operators' Remediation and Diagnostic Interface(GEORDI) offers an alternative to the scenario mentioned above. GEORDI provides a method for the diagnosis, triage and remediation of system problems from the sysadmin's current location. This is accomplished via a commodity PDA and wireless TCP/IP connection. Using existing industry-standard software such as ssh [Ylonen] and sudo [Courtesan], GEORDI creates a secure connection to a remote host without requiring additional host software. This approach minimizes the likelihood of introducing new exploits simply through the use of this access method. Interaction with GEORDI is through the native handheld user interface rather than the more traditional console/keyboard paradigm.

We chose to focus our research in addressing this situation on the PalmOS family of PDAs. They hold a commanding market share over other handheld devices and offer mature development environments across multiple platforms such as Metrowerks' "Code Warrior" for Microsoft Windows and Apple MacIntosh platforms, and the prc-tools GCC toolchain and pilrc resource compiler for many UNIX systems. The general architecture presented should be easily adaptable to other platforms since the software it is based on is available under one or more open source licenses.

An Itch to Scratch

Like many other tools for system administrators, GEORDI came about as a result of the authors trying to scratch an itch. For one, we found that we were still traveling to sites or terminal rooms seeking out remote access for problems that were often trivial to solve in comparison to the effort expended to address such issues. Frequently, we were tasked with driving to work to restart a server or application whose death had been broadcast to our pagers. While emergency maintenance has always been a part of a system administrator's job, we found it increasingly frustrating in light of the growing sophistication of alert devices. Indeed, it was becoming commonplace for system professionals to carry CPU power exceeding that of recent legacy desktops. While it would not be possible to obviate every instance of on-site repair, we theorized that many midnight treks to the server room could be prevented with effective application of the devices we always had with us.

The other facet of this was the desire to carry one's system environment with oneself and to be able to copy it to new locations. Not every situation would involve a fresh disaster; even during normal operations, it would be advantageous to take the current state of a task in progress and move it to another system or device. Desktop layouts, terminal window states, and even programs and scripts in execution should be portable across both large systems with high-speed connectivity, and smaller, less connected ones as well.

Finding significant areas of intersection between these two efforts, we decided to work together on a tool that could serve as a sort of remote "first aid kit." It would let a sysadmin collect data on system health and activity, allowing him or her to make a decision if a trip to the site was warranted or if an issue could be addressed from his or her current location. This would require research in the areas of UI design, connectivity and security as well as investigations into the systems architecture of small, mobile computing platforms.

The initial goals behind GEORDI then were:

- To provide a way to quickly examine and fix system problems from arbitrary urban locations. It would be nice to restart your webserver from the middle of the Gobi Desert, but that kind of scenario is far more dependent on industries and technologies not under our control. It is best to focus our energies on those things more within our grasp.
- To do so using common, commercially available handheld systems such as PDAs, palmtops, smart pagers, etc. With CPU speeds between 33-200 Mhz, 8-64 MB RAM and 16-bit color displays, we now carry on our persons what used to sit on our desks 5-10 years ago. Additionally, most IT departments are unlikely to fund or otherwise be able to justify the purchase of additional single-purpose gadgetry, especially for a large staff.
- To do so in such way that poses no greater security risks than already exist through current remote access methods.
- To do so within the realm of current open source software licenses so that others have the chance to build on our work to suit their own needs.
- To encourage research and debate on what future tools for performing system admininistration tasks should look like, with the focus on a practical, needs-based UI that's as portable as possible. Users and equipment are spreading around the globe and we're expected to keep up with them. This can't always be done from a 17 inch display with a 600 Mhz CPU connected to a T1 in the datacenter.

Applications and usage models

Desktops vs. Handhelds

The idea of remote access from handheld devices is not entirely novel. Indeed, the presence of terminal applications on small keyboard-based organizers can be seen as far back as the early 1990s in the HP95LX palmtop and Sharp Wizard/Zaurus. At the time of this writing, there was even a terminal client and Lynx web browser for the HP 48 calculator [Costar]. Handheld computers themselves go as far back as the early 1980s with the Sharp 2100N, marketed in the US as the Tandy PC-1 Pocket Computer.

Many connectivity options exist for more recent handhelds. Numerous VT100 [Hall], telnet [Ptelnet], ssh [GoldbergSSH] and other tty-like clients are available for most modern PDAs, including the PalmOS family of PDAs and PocketPC. The reasons for this proliferation of terminal clients is rather obvious. Now, as then, the world still largely runs on the command-line when it comes to systems and network management. Graphical status tools may be ideal for displaying status and performance; to actually affect or control a system remotely from an arbitrary location, however, the only reliable constant is a terminal.

While these applications present the sysadmin with the familiar command line interface, any attempt at sustained work will demonstrate that using these tools on a handheld is quite different from using them on a traditional system. The screen is smaller, and the keyboard is often laid out differently, if there is one. The API for the handheld may provide only limited support for text display outside of native forms-based support, resulting in emulation bugs or requiring termcap entries specific to each terminal application and handheld used on the remote host. The PalmOS API, for example, has no concept of an actual console and all characters must be draw on the screen using bitmap coordinates with one of the WinDrawChar() functions [PalmOS]. Scrolling, placement and update are likewise left as an exercise to the individual application.

One approach to dealing with this is the one taken by tools such as VNC [Richardson, et al.] that provide a complete pixel-by-pixel replication of the remote system's screen environment. VNC also provides state-preservation that allows a user to move mid-keystroke from, say, a Sparc desktop to a Windows PC or Mac and pick up typing exactly where he or she left off. Depending on feature support in the client, these events can even be reflected back to the remote system the user just left, providing a useful means of remote support in a heterogeneous environment.

The chief problem here is largely one of bandwidth. VNC was originally designed for use in an environment where the primary network media is

ATM and the effort involved in shipping around chunks of a 1024x768, 24-bit desktop image is comparatively small. This continues to work well over wires at 10/100 Mbit speeds but begins to suffer performance problems once the pipe drops below a T1. The situation worsens when one considers that the maximum speed for the PalmOS serial or IRDA port is 57,600 bps. In informal tests with PalmVNC on a 33 Mhz Visor Platinum, we found approximately a 5-8 second delay between the time one initiated an action on the Visor and the time that action was reflected back to us on the screen. When the connection is reduced to 19,200 bps, a speed reasonable to expect from CDPD or micro-cel carriers, the delay shoots up to 30-48 seconds. There are 802.11 modules available for a number of different PDAs on the market, but their short range limits their usefulness and can necessitate a significant infrastructure investment to provide coverage for a campus environment.

Numerous UNIX vendors and individuals at one time or another have made some effort in non-CLI-based administration interfaces, each meeting with varying degrees of success. We chose to revisit those tools we had encountered previously in our careers,such as the Solaris AdminTool, IRIX System Manager and AIX SMIT. While they were generally focused only on administration of the local host they ran on, each came pre-installed on new systems from their respective vendors by default and each had a different approach to the idea of system admin shells. The Solaris AdminTool focused primarily on the novice user, providing a way for him or her to quickly and easily perform user account management and peripheral/device control. SGI's System Manager extended this somewhat further by building the tool into the normal user desktop menubar and providing greater detail on the status and configuration of disk,network and other peripheral devices. SMIT differed significantly from the other administrative shells. Overall, it was probably the most directly useful to our work; its ability to record keystrokes as one stepped through its menu-driven UI and to convert these into shell scripts provided some of the inspiration for what later became GEORDIs Command Builder.

In each tool, we also encountered a number of deficiencies.

- Their proprietary nature and lack of support for managing systems remotely limited their usefulness, even within the vendors' product families.
- They tended to do one or two things well, but fell significantly short in other areas.
- Some tended to exhibit poor state/error control. Pushing the button and not getting an error dialog didn't always mean things worked. It was often necessary to drop to a shell to confirm the operation occurred as expected.

A subtle yet more crucial drawback in all of these tools was that they were designed for large, general purpose systems with a usage model that involves sitting down and committing to several minutes (or more) at console. It is almost expected that the user won't finish the task or find what he or she is looking for without some digging, so the system is designed to accommodate and, in some ways, encourage this.

Handheld systems in contrast follow a completely different usage model. They are used for seconds at a time as people pick up the phone or dash through the airport. They are tossed in the car and then glanced at furtively for directions while we drive down the road. If a handheld exhibits any sort of "boot time" or forces the user to stop and focus on the unit, rather than the data it contains, it has failed as a useful device. Consequently, the data these devices store is equally brief. IP addresses,dates, passwords, error codes, building numbers and other similar scraps of information fill their memory.

How this affects a system administrator's ability to efficiently do tasks on one of these devices is not immediately apparent until the first time he or she must scribble out something like

```
ps -aux | awk '/luser/ \
  {printf(''killall %s0', $9) }' | sh
```

on a PDA terminal client using the stylus instead of a real keyboard.

This is, as one might guess, far from optimal. What can be done? As system administration frequently requires the ability to interact with a system at its lowest levels, access to the command line is critical, even if it is awkward and inconvenient to use in some situations.

GEORDI Design

UI Design

Initial Efforts

The GEORDI UI began life as something similar to a desktop GUI, but with an emphasis on being highly configurable and extensible. Menu items could be moved around and re-ordered, so that the frequently selected choices could reside on top. Commands and scripts, represented graphically, could be linked together by dropping them on or next to each other. The primary focus here was on modularity and configurability. We felt that we already had one or two strikes against us from a industry cultural perspective simply by virtue of the graphical nature of our tool.

This initial version was passed around the table at local user group meetings, and the responses were largely negative. The UI was too confusing for the available screen real estate and quickly led to users being lost in a maze of icons, tear-off menus, drop-down lists and the like. The one tool they did seem to find and use consistently was the CLI popup, totally defeating the purpose of the tool. In the drive for maximum configurability, we had inadvertently recreated those tools we resented so much on the desktop.

In a number of ways, this turned out to be a blessing in disguise. Writing a Palm app using almost nothing but the Gadget resource involved some rather heavy lifting in the code. As this was only the first revision,things were bound to become worse. We still felt we were on the right track with the "LEGO Brick" model as most of the complaints seemed to focus on screen size and performance issues.

Squeak

Squeak is a modern implementation of Smalltalk-80 [Squeak], one of the original object-oriented programming languages and the progenitor of much of the OO programming movement in the 1980s. An implementation is available for the Compaq iPAQ handheld. With a 200 Mhz StrongARM CPU, 32 MB of memory and a high-resolution color screen, the iPAQ is a considerably more capable system than the Palm. Squeak also boasts a much richer set of UI and data type primitives than PalmOS. Additionally, Squeak's VM architecture offers write-once portability, and the prospect of having usable Geordi clients for any of Squeak's many supported architectures was extremely attractive. A Squeak version of GEORDI similar to the aforementioned "Brick" architecture was fashioned in short order, and a feature was added where one could drop scripts as objects onto a small TTY in the corner of the screen. Unfortunately, this nascent effort was shelved just as quickly as the first Palm implementation; serious bugs in Squeak's graphical interface manager made testing and debugging impossible. These issues led to our decision to concentrate exclusively on the Palm implementation of Geordi for this phase of our work.[1]

A Forms-based UI

We noticed[2] that many of the UNIX commands used for reporting system status and configuration produce output in a row/column format. The PalmOS API, coincidentally, provides extensive support for row/column table forms with a callback handler mechanism allowing users to perform actions on the data in those forms.

By wrapping the output inside PalmOS form elements rather than just writing it out to the screen, we are able to attach event handlers to each table cell, allowing a variety of actions or additional related dialogs or forms to be displayed in response to each tap. Through this we are able to provide a coherent way to execute and display the results of commands such as top(1), df(1), netstat(8), ifconfig(8), etc. In retrospect, this seems quite an obvious approach, but we are not UI experts and there is a cultural bias in our profession that seems to say "If it doesn't have a console mode, it can't be powerful enough to be useful."

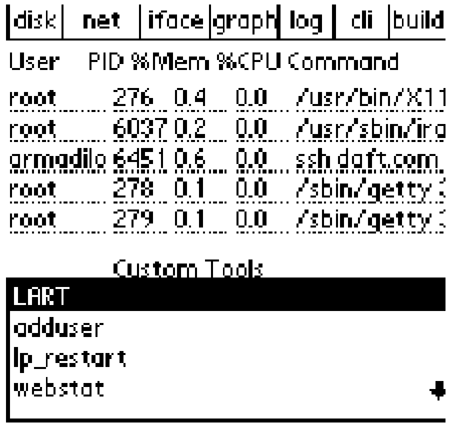

Figure 1: Process listing from remote host.

In Figure 1, we see a process listing from the remote host. The buttons across the top of the screen lead to similar forms, which provide access to other common system tools like those mentioned above. To send a signal to process 278, we tap on the PID column and are presented with the display in Figure 2.

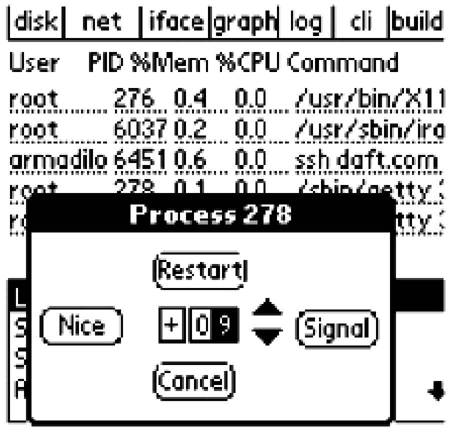

Figure 2: Signaling process 278.

From this dialog, we can easily set the nice value of the process or send it any valid UNIX signal. Similarly, to see what PID 278 is running, we can tap on the ps table column containing the command name to see the full text of the command being executed under that PID.

Finally, if necessary, we can pop up a text form similar to the one in Figure 3 to enter a specific command.

This experience taught us an important lesson in UI design and usability. Good tool design should flow from both a specific need and philosophy where the manifesto of the tool precedes its actual creation. The most successful UIs are those which impart a "Zen of . . ." design on their applications. They fill an actual need, not a marketing goal, and are consistent in form

[1]This has since been addressed and resolved as of August 2001. Returning to this platform is one goal of future work

[2]Actually, it's more like we were knocked on the head with this by our friend and colleague Jim Dennis. After testing yet another version of the Palm GUI, Jim observed that he "just want(ed) a ps table where I can tap on things and kill them!"

and function. The success of the Mac and Palm are very much tied up in this. We feel GEORDI follows this because the forms and other UI elements it uses are the same as those found in the DateBook or PhoneBook. Even command-line tools share this to some extent. -V usually will give a version number, -v verbose output, -q quiet operation a single – to represent standard I/O and so on.

Figure 3: Text box for command.

Communications Issues

There are a number of communications issues that present potential problems for using handheld devices for remote system administration and monitoring. Service availability is a major one. As of this writing there are only a handful of wireless data carriers, and even fewer whose networks provide IP connectivity. Instead, most wireless carriers are based on store-and-forward or other non-persistent, non-stateful technologies. One example of this is the Palm.net service for the Palm VII family of wireless PDAs, which uses the BellSouth pager network. Under this "transactional" model, data requests are sent from the PalmVII to a proxy server which then converts them into proper HTTP requests to retrieve the web page requested by the Palm VII. A separate connection returns the requested data to the PDA.

For those networks that do provide a PPP or PPP-like service on their network, the picture is better, but not by much. Connection speeds range between 9600 and 38,400 bps. For our work, we used the OmniSky service, which is a CDPD network supporting connections up to 19,200 bps.[3] There can be significant delays on the order of 20-60 seconds, or more, to establish a connection to a remote site and then the same or longer during the lifetime of the connection. Often, latency on the wireless part of the connection can exceed the timeouts for a TCP/IP connection, causing it to drop and forcing the user to restart his or her session. This leads to the interesting result where a simple network like Palm.net can end up providing a better overall user experience. Since it does not employ protocols which depend on a continuous connection to function, it appears to the user to be more robust.

The Bandwidth Basement

While networks like Palm.net may not allow for protocols like ssh to be run over them, the bandwidth available is still sufficient for our purposes. The PalmOS Web Clipping Developer's Guide recommends that applications using the Palm.net service should send no more than 70 bytes of payload data per query and receive no more than 360 bytes per response [PalmPQA]. This is after a 5-60% compression ratio of the transmitted data, depending on whether images or text are being sent. Similar restrictions exist for devices like email pagers. At first glance, this might seem too restrictive for our purposes. Closer inspection reveals these limitations to be rigid, but less confining than we are initially led to believe.

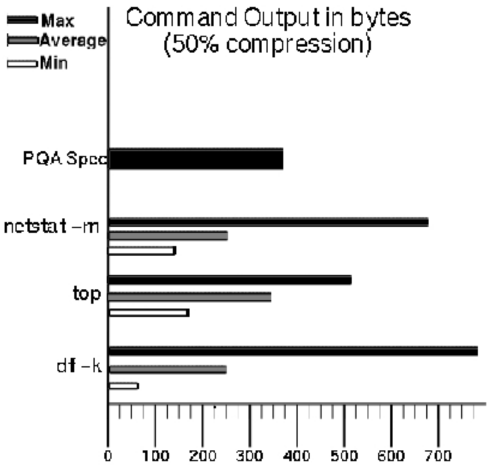

Diagram 1: Transfer sizes.

Here we have a small group of common commands a sysadmin might run to get a sense of the overall health of his or her system and track down potential problems. As can be seen from the graphic, the general trend is that the number of bytes generated by these commands is well within the guidelines recommended by Palm. Even in the maximal cases, none of the data ever reaches even 1KB in size. This gives us a good idea where the basement is in terms of minimum acceptable bandwidth.[4] For faster links, e.g., 9600 bps and above, response time should not be an issue as long as we limit the duration of the connection to what's required to get our work done.

[3]We discount the Metricom Ricochet 128K service as its future is undetermined at this time.

[4]These results were achieved by sending the script

```
df -k|wc -c;top -b -n 1 |
    awk 'NR >6 && NR < 20 {print}' |
    wc -c;netstat -rn | wc -c
```

to several email lists the authors belong to, each with significant subscriber bases of systems professionals, running a variety of Linux,*BSD and Solaris systems and compiling mean, min and max stats on the returned results. No attempt was made to filter out NFS mounts,loopbacks, virtual interfaces, etc. Since this data was purely text-based, we assumed a 50% compression ratio

GEORDI Security

GEORDI Security Basics

With a good idea of the parameters for the raw communications space we were in, we turned our attention to security. We knew that the capability to form an encrypted, preferably authenticated, connection to the remote host would be critical to the success and use of GEORDI. If we could not provide this functionality, the tool would be largely useless save in an academic context of how NOT to implement a GEORDI-like tool. Control of access to privileged commands was equally if not more important, as it is much easier to gain physical control over a handheld system than it is a laptop or desktop. Most privileged remote access is allowed with the assumption that, in addition to whatever electronic security is in effect, the user will still have considerable physical control over the access device. This is not nearly as certain a proposition with handheld systems.

While the PalmOS was a constraint or hindrance in some aspects of this project, there were side benefits to these when it came to security. For a start, the PalmOS is a single-threaded OS which does not provide a shell or user environment one can login to. Limited system memory provides for a very limited number (4 under PalmOS 3.5) of network sockets which must be shared by inbound and outbound connections. The Network Library supports the Berkeley Sockets API, but only through a wrapper layer that is opaque to client and server-side connections. To limit the amount of information available about the systems it contacts, GEORDI does not store usernames, passphrases, keys, etc. in any permanent or temporary databases on the Palm device. These are all held as variables within the program segment itself and are wiped when authentication timeouts occur or when the program exits. This provides protection against them being copied off the PDA through surreptitious beaming [Kingpin] or transferred to another system as a result of a sync operation.

A Daemon-Based Approach

We initially considered a daemon-based approach, where a daemon on the server side would manage client connections and provide both access and privilege management facilities. A GEORDI client would connect to the remote server, present some sort of credentials in the form of a signature or key and request authentication. Upon success, the GEORDI client would be sent a "package" of capabilities outlining what it could do and how long those capabilities lasted.

A Capability would represent a command or series of commands the client was authorized to perform. This would be sent back to the server along with the Key when GEORDI wanted to perform an action there. Upon success, the server would execute the command for the GEORDI client and return the results. The TTL indicated how many times the client could use that capability before it had to request authorization again. Policy indicated whether the key was timestamp or iteration-based. Finally, Server-side Directives gave the server the ability to command and control the client. If a client device was lost or kept trying to perform operations beyond the scope of its privileges, a server directive could be sent to disable the GEORDI Client.

Capability	Key	TTL	Policy	Server–side Directives

Figure 4: Capability packet diagram.

This was ultimately rejected for traffic as well as security reasons. For one, the worst case of a senior-level person needing access to a wide variety of commands could lead to a situation where a storm of capability data would need to be loaded on a GEORDI client at each startup. This might be acceptable on a high-speed wired network, but over a slow wireless link, it would result in significant startup delays. Additionally, this kind of system could expose detailed information about available system tools and other sensitive server configuration details in the capabilities block if the encrypted session was ever compromised. Finally, there was the ever-present danger of attacks on the server daemon itself and possible exploits therein. A general antagonistic attitude in the security and systems community towards new daemons cemented the cons of this approach in our minds, and we decided to look elsewhere.

The ssh *Approach*

We had avoided an interactive solution using something like ssh because of bandwidth concerns. The size of data that would fly back and forth as keys were exchanged and the CPU time needed to do the necessary cryptographic computations was definitely a concern when aiming for a timely connection. Having seen just how much our other approach could consume, we decided to have another look at it. Ian Goldberg, through the TGssh [GoldbergSSH] application, demonstrated that is was possible to implement a functional SSH client on a handheld system – specifically the Palm Pilot Professional – with a rich variety of encryption algorithms such as 3DES, IDEA, RSA, Blowfish, and others. Our implementation hardware was a more advanced Handspring Visor Prism, so it was encouraging to see that ssh had been implemented on such an early model Palm. Tests of TGssh over a serial link and IRDA at speeds between 19,200 and 38,400 proved that such an application was quite usable from a communications standpoint.

At the same time, we were invited to speak at BayLISA[5] on our work up to that point. We performed an informal survey of the audience and found that the

[5] the local LISA chapter for the San Francisco Bay Area

majority considered password-based ssh to be an acceptable tool for performing remote administration; of that majority, about half admitted to regularly ssh-ing in as root. While this certainly does not reflect the most strict and secure remote access procedures, it does provide insight into the where most place themselves along the security/convenience spectrum.

The choice of ssh as an access method also provided additional benefits with regard to user logging and access to privileged commands. Once logged in, we can make use of any access or privilege control mechanisms already on the system. SSU [Thorpe] held some initial attraction for its use of distinct passwords for privileged operations, per-user command configuration, tight coupling to ssh and account-less login. In the end, we felt that sudo was more suited to our purposes. Since our access of systems is solely from the outside by users of potentially varying authority and responsibility, we had to assume a stance of trusting users as little as possible. Since we were already vulnerable to whatever bugs or holes exist in ssh, the ability in sudo to do more detailed logging and run commands as a user other than root tipped the balance in its favor.

General Security Issues with Handheld Systems

Despite the measures we have employed in the security of GEORDI itself, systems like GEORDI introduce some unique threats to the area of system and network security simply by virtue of the devices they run on. The most prominent of these revolve around the mobility and size of the device that runs GEORDI. The worst case scenario is loss of control or removal of a PDA from the hands of an authorized user. Hardware-based authentication proves only that the device is trusted and known to the system and says nothing about the person using it. Precautions must be taken to guard against Bob grabbing Alice's PDA and running off with it. A simple physical tether like the "The Bond" [Force] would help.

Should this still somehow occur, Alice would quickly call the office to have her server passphrase or keys repudiated. A more forward thinking sysadmin would have also installed a screen-locking program such as "LockMe!" [Witte], "JotLoc" or "GridLock" [GridLock] which locks out the device itself after a certain period of time. Access is regained through the entry of a certain passphrase, as in the case of LockMe! or through the reproduction of a certain pattern as used by GridLock. These are analogous to screen-saver lockouts in the desktop world but are more effective on the Palm. Zero-length boot time and lack of access to the underlying OS contribute to this.

The device can be reset, but the reset process is a physical, multi-step operation; it is accomplished by depressing the pinhole reset switch at the back of the device and subsequently depressing one of the scroll buttons. A simple soft reset does not flush alarms or pending system events, so it should also not disable the lockout program. It is possible for the device to be put into a debug mode, under which access via the serial port is possible, but this has to be explicitly enabled via a Grafitti [Grafitti] stroke during normal operation prior to the reset action. As with any security-related tool or application, it is advisable to test several applications before making a selection for production use. A battery of tests to ensure the utility secures the correct resources and works properly on the model and OS version of the PDA you are using is highly recommended.

Other problems occur as a function of the wireless communications technology itself. Encryption only over the wireless part of the trip from the client to the remote host, proprietary protocols and software, and weak or flawed encryption are all common problems when dealing with wireless networks. The latter two are greater issues which affect both corporate wireless LANs and mobile devices.

We attempt to minimize exposure of the remote system by implementing a timeout of three minutes per connection, after which the user is forced back to the initial login screen to re-authenticate. GEORDI originally operated in a connection-per-command mode to force constant key regeneration, but the overhead of 30 seconds-1 minute per connection over the OmniSky significantly reduced the usability of the tool.

We recognize that there are just some things that can't be done on a system the size of PDA. There will be some cases where GEORDI will not be able to provide the necessary remediation capability needed to fix a particular problem. Some of these may involve network outages, or require the use of a tool or command whose input or output GEORDI is unable to properly represent. Currently, GEORDI does not provide support for access to systems which are inside complex network topologies involving multiple firewalls, DMZs and the like. This could most likely be overcome with some sort of "chat script" system or additional prompting during the connection phase for a user to input a one-time password or additional key. There may also be policy-based reasons within an organization which would preclude the use of something like GEORDI.

Implementation Details

Client Implementation

The handheld device itself is a Handspring Visor Prism, although we use the standard PalmOS 3.5 API, ignoring model-specific features such as color. The wireless device is a Novatel Minstrel S CDPD modem using the Expedite chipset (used for many devices in the "Minstrel" family) with connectivity provided by OmniSky.[6]

Systems constraints within the PalmOS architecture preclude the assignment of a static name to any

[6]See www.omnisky.com for more info

particular device. This holds true for built-in devices such as the onboard serial and IRDA port as well as modules such as the Minstrel. Devices are instead identified at the API level by the service they provide and must be polled each time an application wishes to begin using them.

Functionally, this works out well; An application need only make a request for access to the PPP service to make a connection out to the net. We have successfully used this to connect to target hosts via the OmniSky as well as over the built-in IRDA port. Connections on other PalmOS handhelds using other network devices should therefore work similarly.

Security is handled in two parts. Transaction security is provided by the GEORDI client via the encrypted ssh connection. General device security is provided by the PalmOS security application, although @Stake security services [Kingpin] has warned of a vulnerability in this that allows it to be easily cracked. We have therefore augmented our implementation with a third-party application which restricts the use of the handheld itself as mentioned above. A full survey of available solutions is beyond the scope of this paper, but we found the GridLock and JotLoc [PDABusiness] applications suited to the task. Given the wide range of PalmOS PDAs available for sale, test-driving the available offerings is recommended before settling on a particular application for your own use.

Server Implementation

GEORDI is designed to take advantage of existing user management/access facilities as much as possible, thereby freeing the administrator from having to worry about Yet Another Set Of Configuration Files. If an administrator is already happy with a host's ssh and sudo configuration, he or she has all that is required to allow GEORDI-equipped handheld users to access the system. If not, it will be necessary to set up ssh and sudo first. Below is an extract of the more important aspects of the sshd_config and sudoers files we used in our development and testing efforts.

One thing to note about the sudoers file is that we create a separate User_Alias called GEORDI. It is reasonably easy to embed usernames into the forms that make up the GEORDI UI; therefore a responsible party could make and install GEORDI on a group of corporate handhelds and hand them out to their systems staff. Forcing users to login in under specific accounts lets us exploit the features of sudo more fully than would be possible with ordinary user accounts. Taking this to it's logical conclusion, one could go so far as to set up a chroot jail area and populate it only

```
[...]
User_Alias      PRIMARY=armadilo
User_Alias      LOCALSTAFF=benfell,star,torin,gerbil,rmadillo,kylosohr
User_Alias      GEORDI=geordi,geordi2,eastbldg,westbldg,oncall

Runas_Alias     OP=root,operator

Cmnd_Alias      DUMPS=/usr/etc/dump,/usr/etc/rdump,/usr/etc/restore,\
                      /usr/etc/rrestore,/usr/bin/mt
Cmnd_Alias      KILL=/bin/kill
Cmnd_Alias      PROC=/usr/bin/nice,/usr/bin/renice
Cmnd_Alias      PRINTING=/usr/etc/lpc,/usr/ucb/lprm
Cmnd_Alias      SHUTDOWN=/usr/etc/shutdown
Cmnd_Alias      HALT=/usr/etc/halt,/usr/etc/fasthalt
Cmnd_Alias      REBOOT=/sbin/reboot,/usr/etc/fastboot
Cmnd_Alias      SHELLS=/usr/bin/sh,/usr/bin/csh,/usr/bin/ksh,\
                       /usr/local/bin/tcsh,/usr/ucb/rsh
Cmnd_Alias      SU=/usr/bin/su
Cmnd_Alias      VIPW=/usr/etc/vipw,/etc/vipw,/bin/passwd,/usr/sbin/visudo
Cmnd_Alias      INSTALL=/usr/bin/dpkg
[...]
Cmnd_Alias      MODS=/sbin/lsmod,/sbin/insmod,/sbin/rmmod
Cmnd_Alias      VOL=/bin/mount,/bin/umount,/usr/bin/eject

Host_Alias      RKLABS=miyazaki,otomo,shirow,gainax,ghibli
Host_Alias      AREA66=gecko,iguana,chameleon,komodo,basilisk,tokay,gila
Host_Alias      GEORDI=AREA66

# root and users in group wheel can run anything on any machine as any user
root            ALL=(ALL) ALL
%wheel          ALL=(ALL) ALL

PRIMARY         ALL=ALL

LOCALSTAFF      ALL=PASSWD:DUMPS,KILL,PROC,MODS,VOL
GEORDI          AREA66=NOPASSWD:KILL,PROC,VOL,REBOOT
```

Listing 1: sudoers file.

with those commands and devices that those in the GEORDI group needed to use to do their jobs.

```
[...]
ServerKeyBits 1024
LoginGraceTime 180
KeyRegenerationInterval 600
PermitRootLogin no
IgnoreRhosts yes
StrictModes yes
X11Forwarding no
KeepAlive yes
SyslogFacility AUTH
LogLevel INFO
RhostsAuthentication no
RhostsRSAAuthentication no
RSAAuthentication yes
PasswordAuthentication no
PermitEmptyPasswords no
SkeyAuthentication no
[...]
```

Listing 2: sshd_config file.

GEORDI in action

At base, GEORDI is a forms-based UI wrapper for an RSA/DSA-authenticated ssh connection to a remote host running sudo. The actual over-the-wire commands sent by GEORDI to the remote host are the equivalent of "ssh user@host.subdomain.domain sudo $SOME_COMMAND_STRING." There is a built-in three minute usage window for each GEORDI session, during which the ssh connection remains up. This corresponds to the maximum inactivity time allowed by a PalmOS handheld before it is automatically powered off. Even if this timeout is disabled, GEORDI will return to the main authentication screen after the three minute period and drop its current connection (if any) to the remote host.

On startup, GEORDI reads in a database of approximately 60 UNIX commands. These are a mixture of navigational, network, device and user access commands which exist under most UNIX variants and which are not specific to a particular device or OS distribution. At the same time, GEORDI also loads scripts or commands created in previous sessions by the user with the Command Builder. GEORDI itself can be run without these databases, but the Command Builder feature is disabled if they are missing. It is assumed that the user will have their PATH and other environment variables configured properly in order to use these commands. Not every flavor of UNIX stores its commands in the same directories and providing any sort of pathing information violates the GEORDI security model.

Currently, GEORDI requires that the user type in the FQDN or IP address of the remote host at the start of each session. A username and certificate passphrase is also required to attempt connection. This passphrase is intended for an ssh daemon using RSA/DSA certificates on the remote host. We have tried to make the code as modular as possible, with the expectation that users may need to accommodate something like a SecurID challenge/response system or legacy SSH servers.

If the user authenticates successfully, he or she is presented with a screen similar to that shown in Figure 1. From there the listed processes can be manipulated as necessary by tapping on the appropriate control. In addition to the controls and scripts provided on this screen, a user can construct his or her own custom commands through the Command Builder by tapping the "Build" button.

Figure 5: Login page.

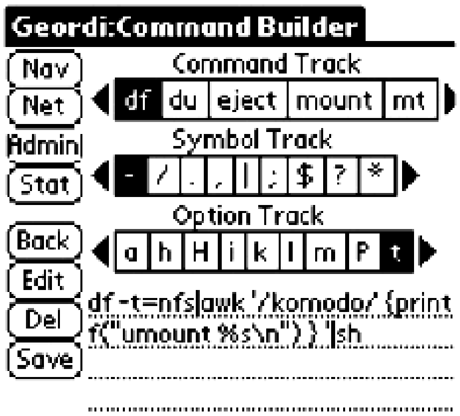

Figure 6: Command builder.

It is here that the aforementioned command bestiary is put to use. New commands or scripts are constructed by tapping on the button containing the desired keyword followed by any additional symbol characters or options in the button rows below the command row. Different sets of keywords are accessed by tapping on the "Nav," "Net," "Admin," or "Stat" commands. Keywords like adduser, mount, df, and the like are organized under "Admin," netstat, ifconfig, route, etc. under "Net" and so on. The option list changes to display only those options for the highlighted command. Saved scripts appear on the drop-down list shown on the main screen.

Configuring the Command Builder

In the version presented here, the Command Builder acquires its data from a companion loader program called GDBI, for GEORDI DataBase Installer. This should be run once for each new GEORDI installation on a particular handheld and then deleted. This approach was taken due to resource constraints in the PalmOS. The program itself is written in C and is easily modified.

The GDBI tool creates a PalmOS record database called ''TrkDB'' which contains the actual commands shown in the Command Builder as well as metadata on that command's relationship to others in the database. This was done to allow for an eventual feature where commands could be repositioned or reordered according to use frequency or user preference. The basic layout of the Builder is a series of ''Tracks'' with commands at the top, followed by regex and punctuation characters in the center with command-relevant option characters along the bottom track. Command sections can be changed along the Command track by tapping on one of the ''Nav,'' ''Admin,'' or ''Net'' buttons off to the left.

Individual records in the TrkDB database have the format in Listing 3.

We expect to have a number of tools available in the near future that will simplify this process down to a CSV-delimited text file suitable for conversion into a PalmOS database, easily installed on the Palm.

What's to stop someone from creating a ''script kiddie'' command database and using GEORDI to wreak havoc? Indeed, not much. However, the conditions under which this kind of attack would succeed are the same as those involved in most other access exploits. Adherence to best practices security measures such as keeping OS and WKS patches current, disabling deprecated or insecure services, and exercising good user account hygiene are the best defenses against this sort of attack. Additionally, while the GEORDI UI makes a good interface for basic triage and remediation of system problems, it's a lousy interface to crack with and is one use case where the attacker would certainly be better off with something like a laptop.

Tools Used

The GEORDI client was built using the PalmOS 3.5 API, prc-tools 2.0 GCC cross-compiler, pilrc 2.7 form description language and Guikachu form designer. These all freely available for downloading off the net at the following respective locations. You will need the first three to build or work on GEORDI. Guikachu is nice to have for forms design.

- PalmOS API and related documentation: http://www.palmos.com/dev
- prc-tools: http://sourceforge.net/projects/prc-tools/
- pilrc: http://www.pilrc.com or http://www.ardiri.com/palm/pilrc
- Guikachu: http://cactus.rulez.org/projects/guikachu/

Outstanding Issues and Future Work

GEORDI in its current form is not perfect, but we feel it represents a reasonable attempt at a remote system administration tool for handhelds. One immediate missing feature is the ability to access systems which require staging across firewalls or DMZs. Additionally, GEORDI is not yet capable of authentication via hardware related tokens or access paradigms which require some sort of human-entered one-time-password to gain access.

The GUI is a good effort for a device like the Palm with a limited forms-based API. Other systems and devices offer a range of features and functionality and we will probably pursue some of our original UI goals on platforms such as the Compaq iPAQ or one of the embedded Linux platforms. In the other direction, we have received interest from colleagues in porting GEORDI to even smaller platforms such as the RIM Blackberry.

There is work to do in the realm of security and authentication, since widespread use of handhelds, wireless networking, wearables and the like challenge the basic notions of what a host is and force us to reevaluate our threat models. The advent of removable media on handheld devices like the Handera 330, Palm m50x and Sony Clie offers some interesting possibilities in regards to the use of hardware tokens or keys to control operation. A number of commercial security software vendors, such as RSA, have either

```
typedef struct {
    int BuildID;       /* Command ID                                     */
    int LeftID;        /* ID of the Build command to the left of BuildID  */
    int RightID;       /* "    "    "      "       "     right of BuildID */
    int trackpos;      /* position w/in the Command Track                */
    int track;         /* Track this command is on                       */
    int section;       /* Section(Nav,Net,Admin) that this command is in  */
    UInt16 CmdTxtLen;  /* Length of the command text                     */
    UInt16 FlagTxtLen;/* Length of the option text                       */
    Char* CmdText;     /* pointer to the command text itself             */
    Char* FlagText;    /* pointer to the flag/option text                */
} BuildCmdObject;
```

Listing 3: TrkDB database record format.

fielded or announced a port of their technology to the Palm and other handheld platforms.

We welcome any and all contributions and comments on this work. We are particularly interested in hearing from those who might see an application for GEORDI on a organizational or enterprise-level scale.

Conclusions

The current array of handheld devices used to alert sysadmins to problems are capable of assisting with solutions as well. We have stated that failure to exploit these resources will leave us at a disadvantage as staff and machines become more mobile and global in scope and location. We have introduced GEORDI, a PDA-based system using existing best-practice remote access and privilege management software, as a first-order example of using handheld systems to aid in the diagnosis, triage and remediation of system problems. We have looked at other remote access applications and administration tools, noting that tools which conform to look and feel specifications for the target platform are generally more efficient than those that impose an external one. The implementation of GEORDI as presented meets many of the needs of today's mobile sysadmin, and will be developed further as interest increases.

Availability

Further information on GEORDI as well as the software can be found at http://www.geordi.org. GEORDI is made available under the GNU General Public License.

Acknowledgements

This work would have been a much paler effort if it had not been for the assistance of a small mob of individuals. Many dedicated colleagues offered criticism and review of GEORDI at all stages, from initial conception through implementation and this paper.

We are particularly grateful to:

- Jim Dennis, Heather Stern, Dave Benfell, Ken Parker, and David M. Zendzian for testing, evaluation and inspiration with the GEORDI prototype(s).
- Craig Latta for his encouragement and insight throughout the life of the GEORDI project, particularly with regards to the Squeak effort.
- Ian Goldberg for graciously answering all of our(sometimes silly) questions about TGssh and pilotSSLeay.
- Strata Rose Chalup, Cindy Fry, Darren Stalder for helping whip the initial extended abstract into shape.
- William Annis and Ozan Yigit, our shepherds, for readings, edits, commentary, and sticking with us right up to the very end of the process.

Author Information

Stephen Okay is a UNIX geek-of-all trades, first exposed to UNIX in 1986 when working as a student consultant at George Mason University, where he obtained his B.A. 1989. Since then he has performed systems support and programming at MITRE's Center for Advanced Aviation System Development, worked on the SRCSS anti-terrorism mission planning prototype for the Sydney Olympics and did a brief stint at Pacific Data Images as a technical director on the animated feature *Shrek*. He currently consults on mobile computing projects as RoadKnight Mobility Labs. He live in San Francisco, California and can be reached at armadilo@daft.com.

Gale Pedowitz is a developer and UNIX system manager, having worked in such halcyon environs as Sony US Research Labs, TenSquare, and eBreviate. Her research interests include dynamic object systems, human-computer interaction, and network security in the context of system administration. A native of New York City, she currently resides in Portola Valley, California and can be reached at gep@ungeek.com.

References

[Ylonen] Ylonen, T., T. Kivinen, M. Saarinen, T. Rinne, S. Lehtinen. "ssh Protocol Architecture," *IETF Internet-Draft draft-ietf-secsh-architecture-07.txt*, http://www1.ietf.org/ids.by.wg/secsh.html.

[Courtesan] Courtesan Consulting, "sudo," http://www.courtesan.com/sudo/history.html.

[Hall] Hall, B., MarkSpace Softworks, "Online 1.6," http://www.markspace.com

[Ptelnet] de Andrade, M. M., "ptelnet," http://netpage.em.com.br/mmand/ptelnet.htm.

[GoldbergSSH] Goldberg, I., "TopGun ssh (TGssh)," http://www.isaac.cs.berkeley.edu/pilot/.

[PalmOS] Bey, C., E. Freeman, D. Mulder, J. Ostrem, "The PalmOS SDK Reference, Document Number 3003-002," p. 33,769, Palm Computing, Inc., 5400 Bayfront Plaza, Santa Clara, CA. 95052.

[Minenko] Minenko, V., Harakan Software, "PalmVNC," http://www.harakan.btinternet.co.uk/PalmVNC/.

[Richardson, et al.] Richardson, T., Q. Stafford-Fraser, K. R. Wood, and A. Hopper, "Virtual Network Computing," *IEEE Internet Computing*, Vol. 2, No. 1, pp. 33-38, Jan/Feb, 1998.

[Squeak] Ingalls, D., T. Kaehler, J. Maloney, S. Wallace, A. Kay, "Back to the future: The Story of Squeak, a Practical Smalltalk Written in Itself," *Proceedings of the 1997 ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications*, Vol. 32, Num. 10, October, pp. 318-326, 1997.

[PalmPQA] Brook, James, Scot Stennis, "Web Clipping Developer's Guide, Document Number 3009-002," p. 28, Palm Computing, Inc. 5400 Bayfront Plaza, Santa Clara, CA, 95052.

[Kingpin] kingpin@atstake.com, "PalmOS Password Retrieval and Decoding," @stake Security Advisory (A092600-1).

[Force] Force Technology, "The Bond: Latch for Palm Connected Organizers," http://www.force.com.

[Witte] Witte, R., "LockMe!" http://wwwipd.ira.uka.de/witte/pilot/lockme/ .

[Thorpe] Thorpe, C., "SSU:Extending ssh for Secure Root Administration," *Proceedings of the Twelfth Systems Administration Conference (LISA 98)*, pp 27-33, Boston, MA, December 6-11, 1998.

[GridLock] PDABusiness Inc., "JotLoc," "GridLock,"http:// www.pdabusiness.com .

[PilotSSLeay] Goldberg, I., "PilotSSLeay 2.01:SSL for Palm Pilots," http://www.isaac.cs.berkeley.edu/pilot/ .

[Stylus] Perlin, K., "Quikwriting: Continuous Stylus-based Text Entry," *Technical Note to the Fourteenth Annual ACM Symposium on User Interface Software and Technology*, November 1-4, 1998 (UIST 98), San Francisco, CA.

[Grafitti] Blinkenstorfer, C. H., "Grafitti," *Pen Computing*, pp. 30-31, January, 1995.

[Costar] Bezemer, J. A., "VT52bis," http://panic.et.tudelft.nl/˜costar/hp48 .