invent

stack buffer overflow protection in hp-ux 11i

november 2001

hp-ux white paper

table of contents

abstract

This white paper describes HP-UX 11i stack buffer overflow protection capabilities and how they can significantly improve the security of HP-UX 11i systems without sacrificing system performance.

introduction

One of the most common types of attack on computer systems is known as a stack buffer overflow attack. By feeding privileged programs an unexpectedly large volume of carefully constructed data, such as unexpected command line arguments, attackers can sometimes trick these programs into executing malicious code that takes advantage of the program's privileges. These attacks can be used to gain unauthorized access to the system, such as a root shell, to destroy or alter data, or to cause denial of service to legitimate users. In many cases these attacks can be remotely launched.

HP-UX 11i can significantly reduce risks from these attacks without incurring measurable performance penalties or requiring source code modification.

security attacks widen

Despite ongoing efforts by software vendors and developers to identify and correct potential buffer overflow problems, all of the major operating system vendors have had embarrassing incidents where buffer overflows in their code have resulted in security breaches. Attacks of this type are especially serious because they can give an attacker full control of a system and because they can often be launched remotely, without the need for a valid account on the system under attack.

Although writing a successful buffer overflow exploit often takes considerable programming skill[1], once written many of these exploits can be launched with point-and-click ease by nearly anyone. Even worse, the attacks can often be easily automated, allowing entire networks of systems to be scanned for vulnerabilities and potentially compromised within seconds. Compromised systems may in turn be used to launch attacks on even more systems.

Experience with the Code Red worm[2] clearly shows the risks of remotely launched buffer overflow attacks. Industry analysts estimate that in just a few days this worm caused billions of dollars of damage[3], despite corporate firewalls and other defensive efforts. Code Red variants laid the groundwork for even more devastating worms, such as Nimda.

Another notable example of a successful buffer overflow attack was the Internet worm of 1988[4] that brought large portions of the Internet down for several days.

These are just two of the more widely known buffer overflow attacks. There are many, many less highly publicized buffer overflow attacks, with new ones being discovered and exploited almost daily.

Buffer overflow attacks have been recognized as a security threat for decades. Before the explosive growth of the Internet, the risks from them were relatively small and tended to be largely ignored except in the most sensitive environments. With the recent growth of the Internet and the increased reliance upon computers for critical tasks, we can no longer afford to ignore these risks.

[1] http://www.securityfocus.com/data/library/P49-14.txt
[2] http://www.cert.org/advisories/CA-2001-19.html
[3] http://www.idg.net/english/crd_code_red_665795.html
[4] ftp://ftp.cs.purdue.edu/pub/reports/TR823.PS

hp-ux solution

Numerous approaches to defending against buffer overflow attacks have been tried over the years, with limited success.

Keeping current on software patches, particularly those flagged as especially security-relevant, helps defend against many known security problems. Tools such as HP's Security Patch Checker[5] help make it easier to identify many of the relevant security patches for HP-UX.

Although software patches can protect against many known stack buffer overflow problems, there is always a risk that new problems will be found and exploited before relevant patches can be created and distributed—or that not all of the required patches will be installed quickly enough on potentially affected systems.

Intrusion detection systems can help detect some buffer overflow attacks, but they may consume significant resources in order to do so, particularly on heavily loaded configurations. Much like antivirus software, they may also require ongoing maintenance to update attack pattern definition files.

Careful design and review of source code is one of the best theoretical solutions, but is not fully practical in the real world, due to both the cost of scrutinizing potentially tens of millions of lines of constantly changing code and the fact that much of the code in use today comes from third parties.

Some programming languages, often with help from their compilers and related tools, include features that make it easier to detect and/or prevent buffer overflows. However, these same programming languages also can impose an unacceptably large performance penalty as well as introduce compatibility issues. For better or worse, the C programming language, in which the vast majority of UNIX® code is written, is not one of these languages.

Recognizing the practical difficulties of these solutions while also acknowledging the very real need to address the stack buffer overflow attack problem at a global level, the HP-UX security team took a different approach.

HP-UX 11i includes stack buffer overflow protection, a new feature that uses a combination of highly efficient software and existing memory management hardware to protect against both known and unknown stack buffer overflow attacks. By removing execute permission from a program's stack pages, it detects the attempted execution of malicious code on a program's stack and prevents the attack before a single malicious instruction can be executed.

key benefits

HP-UX stack buffer overflow protection, new in HP-UX 11i, provides a number of key benefits.

Robust, low-maintenance protection
There is no need to modify the protection code as new variants of stack buffer overflow attacks are created. While a virus scanner typically needs almost constant updates with new virus definition data, HP-UX 11i stack buffer overflow protection never needs updating. As long as the attack causes malicious execution of code illegally placed on a program's stack, which is by far one of the most common attack modes today, the attack will be detected and stopped before even one instruction of code can be executed, regardless of system load or whether the specific attack variant was previously known. The offense is logged and the offending program is terminated before any of the malicious code can be executed. By comparison, intrusion detection systems may lag significantly, especially under high loads, and may only detect the intrusion after the damage has already been done.

[5] http://www.software.hp.com/cgi-bin/swdepot_parser.cgi/cgi/displayProductInfo.pl?productNumber=B6834AA

Industry-leading performance and scalability

HP-UX stack buffer overflow protection has been carefully designed to minimize performance impact on legitimate applications. The only performance penalty is a few microseconds at application launch. By comparison, intrusion detection systems or enhanced security libraries such as *libsafe*[6] can increase program execution time substantially, depending on load, configuration, and nature of the application.

Once an application starts, no system resources are consumed in order to provide protection from stack buffer overflow attacks. From the smallest workstations to the largest, most heavily loaded servers, the incremental cost in CPU cycles and memory usage is virtually zero.

Transparency and compatibility

There is no need to modify a program's code to get stack buffer overflow protection, unlike other products that require time-consuming program modifications, recompilation, or relinking. In some cases this may not even be possible since source code, libraries, and/or the necessary compilers aren't always available when needed.

Well over 99% of HP-UX applications never attempt to execute code from their stacks and need no modification to benefit from HP-UX stack buffer overflow protection. However, a few programs, notably some simulators or interpreters and programs using some older versions of Java™, have a legitimate need to execute code from their stack(s).

Authors or installers of these programs can use the *+es enable* option of *chatr*(1) to flag these programs as having a legitimate need to execute from their stacks. This override mechanism is similar to the zone bypass feature on a home burglar alarm system. The alarm system can remain fully armed even if it is necessary to disable it for a small area.

Trial mode

Many security features are underutilized due to fear that enabling them may have unintended adverse effects on a mission-critical application. HP-UX stack buffer overflow protection provides a "trial mode" that can be used to gain confidence that it will not interfere with legitimate applications. In this mode, potential buffer overflow attacks are flagged as non-fatal warnings instead of as fatal application errors. By running production workloads in this trial mode for a period of time, any affected applications can be more easily identified and corrected.

enabling stack buffer overflow protection

HP-UX stack buffer overflow protection is controlled in two ways. The first control, a kernel tunable parameter, *executable_stack*, is a "master switch" that globally enables or disables stack buffer overflow protection. This can be changed with *sam*(1M) or *kmtune*(1M).

The second control, set via the *+es* option of *chatr*(1), enables fine-grained control for individual binaries, allowing increased compatibility without sacrificing security for the system as a whole. Refer to the "zone bypass" section below and the *chatr*(1) manual page for additional information on this control.

Combinations of these two controls permit site-specific trade-offs between security and compatibility.

[6] http://www.avayalabs.com/project/libsafe/index.html

The possible settings for `executable_stack` are as follows:

executable_stack = 0

A setting of 0 causes stacks to be non-executable and is strongly preferred from a security perspective. When this setting is chosen and a program attempts to execute code from its stack(s), the HP-UX kernel will immediately terminate the program (send it a *SIGKILL* signal) and log (through *dmesg* or *syslog*) the apparent stack buffer overflow attack. It will also notify the IDS/9000 intrusion detection system[7], which may take other actions such as paging a system administrator. The message logged by the kernel is:

```
WARNING: UID # may have attempted a buffer overflow
attack. PID # (program_name) has been terminated. See
the '+es enable' option of chatr(1).
```

If this message appears, check with the program's owner to determine whether this program is legitimately executing code from its stack. If it is, you can use the "zone bypass" method described below to make the program functional again. If you are not certain the program is legitimately executing code from its stack, you should suspect malicious activity and ask the program's owner and your site security personnel to investigate.

executable_stack = 1

(default)

A setting of 1 (the default value) causes all program stacks to be executable. This is safest from a compatibility perspective but is the least secure setting for this parameter.

Note: In future releases of HP-UX, the default is expected to change to 0. Application developers should therefore test their code with *executable_stack* set to 0.

executable_stack = 2

A setting of 2 is equivalent to a setting of 0 except that it gives non-fatal warnings instead of terminating a process that is trying to execute from its stack. This setting may be thought of as a "trial mode."

This "trial mode" enables a user to activate protection in a mode that identifies possible problems but does not alter program behavior. This is helpful when a system administrator wants to enable the protection but is concerned that doing so may disrupt a mission-critical application. By enabling the buffer overflow protection's "trial mode" and running a representative workload, a system administrator can readily identify those applications that would be affected—without altering their actual behavior or risking unplanned down time. After successfully running with this setting for a period of time, system administrators should gain the confidence to fully enable the protection by using a setting of 0.

Note that this "trial mode" identifies *but does not prevent* malicious stack buffer overflow attacks. It should therefore be used only long enough to identify and resolve any issues with legitimate applications.

[7] http://www.hp.com/security/products/ids

“zone bypass” feature

Although over 99% of legitimate HP-UX applications are not affected by HP-UX stack buffer overflow protection, a few programs may legitimately need to execute code from their stack(s). Common examples of these include some simulators or interpreters, or some programs based on older versions of Java. Legitimate use of these programs may generate reports of possible stack buffer overflow attacks.

Competing operating systems have no provision for dealing with this situation. If even one legitimate application must execute code from its stack, then either the stack buffer overflow protection must be entirely disabled or the application source code must be modified.

HP-UX provides a much better solution, similar to the “zone bypass” feature found on home alarm systems. Application owners may mark their binaries as having a legitimate need to execute code located on their stack(s). Programs so marked are exempt from the HP-UX stack buffer overflow protection. This allows these programs to execute correctly without affecting protection of the rest of the system.

This feature should be used sparingly, and only after careful analysis of the application in question. For example, encountering a Java-based application trying to execute code from its stack(s) often means that the binary in question is sufficiently old that it is missing some key security-related fixes to the Java virtual machine, described in previous HP-UX security bulletins. Instead of using the “zone bypass” feature to allow this program to execute code from its stack, it would be far better to get a more current version of the program, which would both address some known security issues and usually permit HP-UX stack buffer overflow protection to be enabled for that application.

To enable this “zone bypass” feature on an application binary, execute

```
$ /usr/bin/chatr +es enable /my/application/pathname
```

This method allows a restrictive system default without preventing legitimate programs from executing code on their stack(s). Ideally this option should be set (if needed) by the program’s provider in order to minimize the need for manual intervention by each site installing the program. However, it may also be set after program installation. (Note that doing this after program installation may cause spurious checksum errors from tools such as *swverify*.)

The “zone bypass” feature for a binary may be disabled by executing

```
$ /usr/bin/chatr +es default /my/application/pathname
```

This is equivalent to the original mode of the binary when created by the HP-UX linker.

Note that these *chatr* settings are always subordinate to the setting of the *executable_stack* kernel tunable parameter. They have no effect at all if *executable_stack* is set to 1. (You can use a zone bypass feature to disable a zone for an alarm, but this does nothing if the entire alarm system is turned off.)

troubleshooting

The vast majority of legitimate applications should work correctly with any of the settings described above, without the need for special treatment. Please refer to the table below for troubleshooting tips.

Problem	**Recommended Action**
The following message appears in dmesg or syslog output, and an application is unexpectedly terminated: WARNING: UID # may have attempted a buffer overflow attack. PID # (*program_name*) has been terminated. See the *+es enable* option of *chatr(*1).	An application has attempted to execute instructions located on its stack. This may be legitimate but could also be a sign of a possible buffer overflow attack. Consult the system administrator or owner of the application for advice. If this message occurs during known legitimate execution of the application, you can run *chatr +es enable* on the application binary to mark it as having a legitimate need to execute instructions from its stack. Ideally this should be done as needed by the application creator, before distribution, but it can also be done after the application is installed. This *chatr* command disables the stack buffer overflow checking for this single binary, but allows the protection to remain active on the rest of the system. If this program does not have a legitimate need to execute instructions on its stack, or if you are unsure of the need, refer the matter to your computer security personnel for investigation of potentially malicious behavior. **Note: See special instructions below regarding Java-based applications.**
A Java-based application tries to execute from its stack, resulting in the error message shown above.	Although you could run *chatr +es enable* on the affected binary to restore correct functionality to this binary, you should first verify that your system has all of the relevant security patches installed. Older versions of Java have known security issues that have been addressed by security bulletins and related patches. The security patch check tool may be helpful in determining the appropriate patches to install. As a side effect of installing these Java virtual machine patches, the applications will usually no longer execute code from their stacks, and thus will not need to have *chatr +es enable* run on them.
chatr(error): cannot open file for writing <pathname>	You may not run *chatr* on a file unless you have write access to that file. Contact the owner of the file or a superuser.

chatr(error): segment HP_STACK does not exist in <pathname>	You have attempted to run *chatr +es* on an old ELF-64 binary. ELF-64 binaries linked with older versions of the HP-UX loader do not contain all of the necessary section headers to permit use of *chatr +es*. Relinking the application with a current version of the HP-UX loader will create the necessary section headers in the executable file, permitting subsequent use of *chatr +es*. This situation should very rarely arise. Very few HP-UX binaries are currently 64-bit, and even fewer of them are privileged. For compatibility reasons, these older ELF-64 binaries are always treated as having executable stacks.

conclusion

No single solution can prevent all buffer overflow attacks. However, the stack buffer overflow protection provided in HP-UX 11i is extremely efficient, virtually transparent, and highly effective at detecting and neutralizing stack buffer overflow attacks, which are among the most common attacks today. This protection can be a valuable component of your overall security strategy.

feedback

HP welcomes your feedback on HP-UX stack buffer overflow protection and its documentation. If you encounter legitimate applications that do not work correctly when this feature is enabled, or if you have comments, suggestions or questions, please e-mail **overflow@hpuxmail.cup.hp.com**.