Chapter XI

Experiences Enhancing Open Source Security in the POSSE Project

Jonathan M. Smith, University of Pennsylvania, USA

Michael B. Greenwald, University of Pennsylvania, USA

Sotiris Ioannidis, University of Pennsylvania, USA

Angelos D. Keromytis, Columbia University, USA

Ben Laurie, AL Digital, Ltd., USA

Douglas Maughan, Defense Advanced Research Projects Agency, USA

Dale Rahn, University of Pennsylvania, USA

Jason Wright, University of Pennsylvania, USA

ABSTRACT

This chapter reports on our experiences with POSSE, a project studying "Portable Open Source Security Elements" as part of the larger DARPA effort on Composable High Assurance Trusted Systems. We describe the organization created to manage POSSE and the significant acceleration in producing widely used secure software that has resulted. POSSE's two main goals were, first, to increase security in open

Copyright © 2005, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

source systems and, second, to more broadly disseminate security knowledge, "best practices," and working code that reflects these practices. POSSE achieved these goals through careful study of systems ("audit") and starting from a well-positioned technology base (OpenBSD). We hope to illustrate the advantages of applying OpenBSD-style methodology to secure, open-source projects, and the pitfalls of melding multiple open-source efforts in a single project.

INTRODUCTION

Posse - *a group of people summoned by a sheriff to aid in law enforcement.*

A variety of reasons, ranging from marketplace ignorance to a perceived trade-off between usability and security, have driven modern operating systems into the undesirable role of a potential lever with which system security can be breached. The use of *any* common operating system platform across an organization can make this lever effective, independent of the organization, its security policy, and security practices.

This problem has been exacerbated by the commercial success of the Internet over the last decade, as the Internet's "end-to-end" (Clark, 1988; Saltzer, Reed, & Clark, 1984) design implicitly relies on host security as the basis of security for the overall system. An example of this reliance and its consequence is the advent of Distributed Denial of Service (DDoS) attacks, effected by multiple computers bombarding one or more target hosts with traffic and disabling these targets.

As the commercial marketplace, and to a large degree the government marketplace, have converged towards a common platform (the dominant commercial operating system, Microsoft Windows), these organizations increasingly rely on the platform to be trustworthy, whether it is so or not. Further, the use of the Internet and computer systems in the functions of all of these organizations has made systems software, as a whole, "critical infrastructure." At the same time, a single point of vulnerability and failure has been created for systems dependent on this software.

The Open Source Alternative

Concurrent with the growth of the Internet, an alternative software development paradigm began emerging. This paradigm had roots in the research UNIX community and its USENET, with some philosophical roots later added with the "Free Software" principles of Stallman. The mid-1960s MULTICS (Daley & Dennis, 1968; Organick, 1972) project, part of the U.S. Defense Advanced Research Projects Agency (DARPA)-supported Project MAC (Fano & David, 1965) at MIT, gave rise to the original UNIX system (Ritchie & Thompson, 1974, 1978; Thompson, 1978) (the name UNIX is in fact a pun on MULTICS) as a reaction to MULTICS system complexity. Unfortunately, in rejecting much of MULTICS, the UNIX system was not able to avail itself of the extensive effort devoted to developing protection

Copyright © 2005, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

models and security kernels (Schroeder, 1975; Schroder, Clark, & Saltzer, 1977) for MULTICS. McKusick, Bostic, Karels, and Quarterman (1996) provide historical details on the emergence of UNIX.

UNIX, as an important consequence of its university base, boasted platform portability of much of the software and easy availability. These, in turn, meant that UNIX became the dominant platform for experimental operating systems research, and the availability of several good books explaining the system internals (Bach, 1986; Lions, 1977a, 1977b; McKusick et al., 1996) meant that the system could be taught. The result, entering the 1990s, was a substantial number of people who understood the ins and outs of most of the operating system. Thus, as the PC became the dominant platform in the mid-1990s, UNIX became the dominant model for "open source" operating systems projects, where system source was fully available for examination and modification. The dominant commercial platform, Microsoft's Windows, is not UNIX based; it has accreted (Cusumano & Selby, 1997) features and technologies starting with a simple microcomputer software platform.

UNIX-based platforms have presumed "shared use" since their inception, were early platforms for network software deployment and refinement, have sizeable and talented user communities, and are available to all for scrutiny. There is a belief in this community (Raymond, 1999) that "many eyes" lead to faster discovery and repair of flaws in software. While "open source" *enables* scrutiny (Raymond, 1999), it does not *cause* it.

The following (quoted with permission) was posted to the "Robust Open Source" mailing list by Peter Gutmann:

I can provide a data point on this based on a disk encryption device driver I wrote about 8-9 years ago. For various reasons too boring to go into here, I never released the source code (AFAIK it's the only thing I've ever written where I haven't published the source). At various times I'd get people sending me mail asking me why I hadn't released the code so it could be reviewed. When I offered to send it to them, they replied that they didn't want to review it themselves, they expected someone else to review it for them. That is, even the people who went so far as to express an interest in the source code admitted they'd never look at it (and furthermore that they'd be quite happy to have some complete stranger tell them it was OK based on the claim that they'd reviewed it)... As an experiment I also planted a comment which should raise eyebrows in some code I released years ago and which is fairly widely used just to see if I'd get any reaction from anyone . . . No one has ever asked me about this, from which I assume that no one's ever looked at the code they're using. That's kind of scary, because the comment isn't in there just to annoy people, you really could build a rather nasty backdoor in there. There may actually be products out there which are released in binary-only form where the vendor has built in a backdoor at that point, although I saw a posting from foo@anon.org in alt.2600 saying he'd looked at the product and it was fine, so it must be OK."

Copyright © 2005, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

That is, many eyes do not help if they are all looking at something else.

The most important contribution, therefore, is the fact that discoveries are shared and can, in some domains (such as networking code), influence commercial code whether these influences are visible or not.

The Marketplace

Concurrent with the emergence of open source has been a drive by some portions of the U.S. Government (notably the U.S. Department of Defense) to develop and/or procure a "trusted" operating system. A major problem with modified commercial operating systems has been the difference in priorities between the marketplace and a knowledgeable, specialized consumer such as the U.S. Government. In particular, the security features and development processes and documentation required have resulted, when the vendors have been engaged, in multiple development efforts—one driven by commercial considerations and the other(s) driven by specific considerations such as security, an audit process, etc.

Separate development of the secure version inevitably results in a **TOAD** (**T**echnically **O**bsolete **A**t **D**elivery) version of the operating system, since the audit process, among other factors, inhibits introduction of new features while underway. The obvious and only cost-effective way to solve these problems is to ensure that no separation occurs, requiring that security considerations be "mainstreamed."

As open source systems are developed by volunteers and often driven by aesthetics (such as a desire for a "secure" system) rather than market considerations, a potential opportunity was identified by author Douglas Maughan of the U.S. Defense Advanced Research Projects Agency and embodied in a smallish (by DARPA standards) program called Composable High Assurance Trusted Systems (CHATS). The goal, at a high level, is to introduce required security features into open source operating systems such as Linux, FreeBSD, and OpenBSD such that they will be in whatever mainstream version exists and that they will be present in commercially supported versions of these operating systems, allowing their procurement by governments and other interested parties.

The initial goals of the DARPA Composable High Assurance Trusted Systems program included adding new security functionality to existing open source operating systems, as well as the political/community effect of demonstrating the value of useful security and analysis tools and techniques to the open source community. This approach by DARPA to work "directly" with the open source community was seen as a risky endeavor by both parties. The open source community was leery of DARPA's commitment to open source, and DARPA was unsure of this new role of research partner and the uncertainty of product delivery. However, DARPA felt that these open source technologies are critical for systems of the future to be protected from imminent attack. The CHATS program has focused on developing the tools and technology that enable core information infrastructure systems and network services to protect themselves from the introduction and execution of malicious

Copyright © 2005, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

code and other attack techniques and methods (Sullivan & Dubik, 1994). These tools and technologies are intended to provide the high assurance trusted operating systems need to achieve comprehensive, secure, highly distributed, mission critical information systems. The CHATS program intended to fundamentally change the existing approach to development and acquisition of high assurance trusted operating systems technology by dramatically improving the state of assurance in current open source operating systems and, further, developing an architectural framework for future trusted operating systems. Such technologies have broad applicability to many programs within DARPA and the DoD (MITRE, 2003).

A most important consequence of the CHATS approach is that technologies developed under the program are demonstrated and evaluated on a large number of open source system platforms, for all to see and use. The open source development model provides a conduit for technology transition directly into products and services that will employ and support trusted operating system technology.

POSSE: TOWARD AN OPEN SOURCE SECURITY COMMUNITY

The Portable Open Source Security Elements (POSSE) Project at the University of Pennsylvania is an example of a DARPA Composable High Assurance Trusted Systems (CHATS) project. In this section, we will describe the goals of the POSSE project (such as supporting widespread availability of high quality cryptographic systems) and the project organization we have used to accomplish these goals. The project organization has generally worked, although several challenges have arisen over time. Nonetheless, as we detail here, the project has been successful both in its internal goals and in its goals of influencing both other open source projects and commercial vendors.

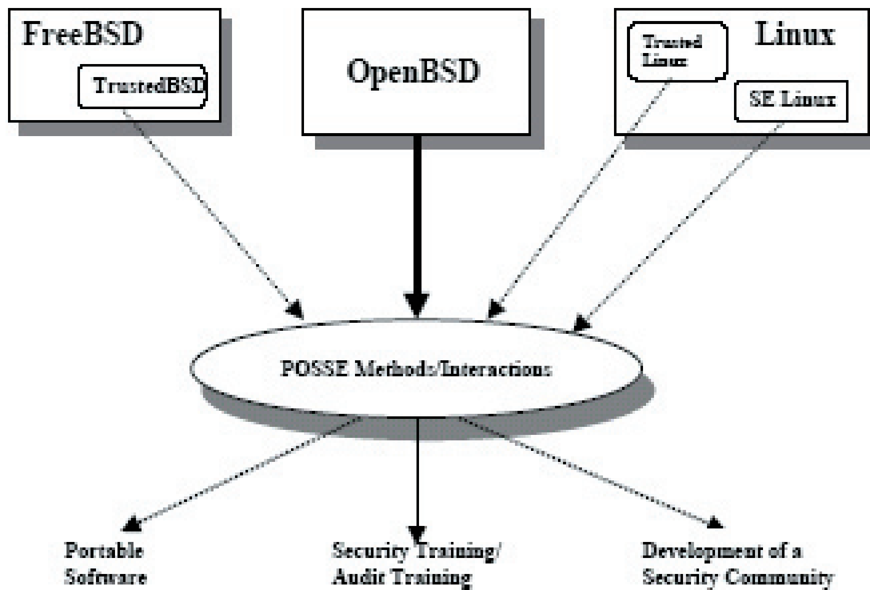
A major goal of POSSE is the development of a (growing) community of individuals interested in and capable of enhancing the security of operating systems. Open source systems serve three purposes towards achieving this goal:

1. They provide a natural diversity, avoiding the "single point of failure" noted above.
2. They provide a basis through which a community of developers can express their knowledge about secure systems.
3. The "open source" characteristic of the software allows the knowledge to be freely shared, even with those who might not themselves choose to share knowledge.

Our model is illustrated in Figure 1. What the model shows is that the POSSE project not only generates its own portable security technologies, but takes a stronger social engineering stance than the "chuck wagon" approach of putting the technolo-

Copyright © 2005, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Figure 1: The POSSE Synchronize and Synthesize Process Model.



gies out and shouting "come and get it." Rather, meetings of developers (at the "waist" of the diagram) build up the strengths of the security community, cutting across project boundaries, and raise all boats on the same tide.

POSSE Project Goals

An abstract view of the overarching project goal is to create and grow a community of open source developers with security as a major focus. Without getting into debates of software engineering "religion," our team studied open source operating projects and found that the OpenBSD project had many of the properties we desired. In particular, it had a very strong focus on security issues, had a small but extremely capable group of developers—several of whom were extremely interested in the technical contributions we wanted to make—and the project leader, Theo de Raadt, was interested in the basic proposition of community building.

Much of the project focus beyond the technological developments has, in fact, been on community building. Important sub-goals have been:

a. Propagating technologies such as the OpenSSH secure shell, which is now distributed with, among other platforms, the Apple Macintosh OS-X, as well as maintaining the multiplatform portability of the OpenBSD system itself (see OpenBSD.org).

Copyright © 2005, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

b. Exporting methodologies such as OpenBSD audit to multi-OS security infrastructures such as OpenSSL, and investigating the strength of tool-based versus expert audit in this task.
c. Collaborating with other open source and free software efforts on security projects of common interest, such as an Extended Attribute File System with TrustedBSD (part of FreeBSD), an open source secure bootstrap with the University of Maryland, and an IPSEC for Linux (Keromytis, Ioannidis, & Smith, 1997).
d. Large face-to-face developer meetings, typically before or after major conferences that attract developers such as USENIX. These meetings have proven surprisingly successful, resulting in, for example, a new packet-filtering firewall for OpenBSD, called "pf."
e. Collaborating with security hardware vendors to rapidly generate support software for their devices, such as cryptographic acceleration hardware.

While we will say more in the section in this chapter on POSSE outcomes, in the Spring of 2003 we feel that, on the whole, these goals have been and continue to be met.

POSSE Project Organization

One of the first questions is how one would organize such a project. While the usual challenges of distributed organizations were all present (decision-making, personnel changes, control of resources, etc.) some particular challenges we faced were raised by the combination of goals and the fact that the CHATS program was funded by DARPA, an agency that is part of the United States Department of Defense.

a. Many of the OpenBSD volunteers were working on their own time but were employed by commercial enterprises.
b. The work we envisioned for POSSE demanded essentially full-time commitments for the OpenBSD and OpenSSL developers responsible for certain sub-projects.
c. Many of the OpenBSD and OpenSSL participants are non-U.S. nationals.
d. Open Source projects do not have a corporate or non-profit corporate structure with which contracts can be negotiated.

We have worked out a solution that has largely been successful. The University of Pennsylvania has contracted to DARPA to perform the items in a statement of work more or less covering the POSSE goals, with some more details as laid out here in the section on POSSE Project Management Challenges. Several U.S.-based, OpenBSD developers became Penn employees. Subcontracts were used for one other U.S.-based developer, and subcontracts were created for Columbia University, as well as subcontracts in Canada and the U.K.

Copyright © 2005, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

The University provides an ideal structure with which to carry out such arrangements, since it is a U.S. entity with a structure capable of contracting, has many modes and methods for employing and contracting, and has intellectual property policies for software that are extremely attractive for a funded open source project. DARPA's only request has been an acknowledgement that DARPA funding was used to create the software; the BSD license rights are completely preserved. It is interesting to note how frequently DARPA is acknowledged in the OpenBSD source tree—many of the acknowledgments are in the original Berkeley source, but more and more (53 in OpenBSD 3.3) are showing the POSSE agreement number!

Our project takes a broader view of what we must do than technology alone. We see that the important tech transition is first among the small number of individuals in each open source effort who are security-focused and second among the core teams of each effort. While these groups are one and the same in the OpenBSD effort, and it is unique in this respect, the important intellectual "customers" are developers who should have their "security" thinking caps stapled to their "developer" thinking caps, so that security is a first-class consideration in every open source effort. Our effort to document the OpenSSL auditing process, to involve many people in development activities, and our aggressive outreach to other projects, enabled by the DARPA resources, raised everyone's standards by several notches.

POSSE Project Management Challenges and Solutions

We outline here four major challenges we faced and our approaches.

1. **Decentralized development.** The OpenBSD and OpenSSL development communities are worldwide and mainly volunteer. POSSE hired two developers (authors Rahn and Wright) at Penn as senior software engineers, residing in the Midwest and Middle Atlantic regions of the U.S., and structured a subcontract with AL Digital, Ltd., a UK firm through which Ben Laurie's services (Laurie is an OpenSSL developer and an author of this chapter) were made available. Such geographic distribution means there must be good communication channels (for example, Internet Relay Chat and Instant Messaging), people must be familiar with and trust each other (frequent communication for trivial matters can be annoying), and tasks must be neatly separated so people can work independently as much as possible.
2. **Integration with existing working methods.** There are already cultural mechanisms and protocols to build consensus among members of the developer community. These mechanisms and protocols can be leveraged by using developers who are already aware of the processes and culture (e.g., Keromytis, Rahn, and Wright), although a certainly degree of friction will always occur because of potentially conflicting goals—this is the overhead of developing a consensus.

Copyright © 2005, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

3. **Minimize administrative overheads.** We used the structure and specialized skills effectively. In particular, the University has significant resources for purchasing, sub-contracting, and reporting. As academics must typically both perform and report on research, it was natural for the academics on the project (Smith, Greenwald, and Keromytis) to write quarterly reports, aggressively report on technical progress in the academic literature, and inject scientific rigor where appropriate. This had the benefit of focusing the developer's attention on development.
4. **"Light-touch" management.** From the start of POSSE, we worked very hard to identify capable and highly motivated people and gave them interesting problems to work on. Not surprisingly, they have implemented clever solutions with a great degree of autonomy.

Management of the project, as anticipated, has been challenging. As we noted in the original POSSE proposal,[1] there are the challenges of distributed management, strong personalities, and knitting together of sometimes quite distinct development cultures. For example, the OpenSSL system must work across many operating systems and its collaboration is much looser, less structured, etc., than the OpenBSD development team, which is tightly integrated and led by Theo de Raadt. In some ways, the development culture of OpenBSD resembles the "Surgical Team" model of hierarchy developed by Brooks (1975), while the OpenSSL development model is more analogous to the "Programming Group" model of Weinberg (1974). The OpenBSD methodology is driven by biannual releases that incorporate whatever software is ready for "prime time," while the OpenSSL releases are more event-driven than periodic release-driven. Thus, the OpenBSD model for what OpenSSL should look like and when it should look that way is clear, while achieving larger scale consensus for OpenSSL took more time, leading to some tension.

In particular, one focus of our work had been the support of hardware cryptographic acceleration, as discussed in the next section, and, further, its integration with SSL to accelerate use of cryptography. Our belief was that cryptographic operations should be perceived by users to be fast (as we have recounted elsewhere—see Miltchev, Ioannidis, and Keromytis, 2002, for example), as this would encourage their use. OpenSSL modifications were necessary to accommodate some of these changes and, based on discussions at an early developer meeting, these changes were undertaken by the OpenSSL community. However, the pace and development style of the two teams clashed, as the OpenSSL release and consensus model did not mesh smoothly with the aggressive release cycles of OpenBSD, and some tempers flared, with many telephone exchanges to and from University people acting as intermediaries.

The seriousness of the culture clash should not be underestimated, and dealing with such potential clashes must be dealt with in any management plan intending to meld multiple open source projects. During the lifetime of the POSSE project,

Copyright © 2005, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

an unhealthy and somewhat permanent rift opened up between the OpenBSD and OpenSSL communities. Major management effort was required to prevent a "fork" of OpenSSL (one for OpenBSD and one for the rest of the world), and this effort was and continues to be successful. Within the POSSE project, this rift was smoothed by project-level successes. These included the many OpenSSL fixes, patches, and enhancements that have emerged from both the OpenSSL auditing efforts and the OpenBSD cryptographic framework as well as cryptographic accelerator support, and modifications to OpenSSL to run on OpenBSD. These features are discussed next.

POSSE: OUTCOMES AND SUCCESS EXAMPLES

The next three sections provide examples of the progress made as a result of the POSSE project. The first of these covers the cryptographic framework, the basis of hardware cryptography support in OpenSSL. The second covers the extended attribute file system, intended to provide controls similar to those of security enhanced Linux (www.nsa.gov/selinux). The third covers audit of the OpenSSL system and some of the important consequences for Internet security.

Hardware Cryptography Support

The OpenBSD cryptographic framework (OCF) (Keromytis, Wright, & de Raadt, 2003) uses a service virtualization model that provides access to cryptographic services while hiding details of specific cryptographic hardware accelerator cards (cryptographic providers) behind a kernel-internal API. User-level applications such as the OpenSSL library or the SSH daemon can access the hardware through the */dev/crypto* device, which acts as another kernel application to the framework. While the implementation details of the framework are outside the scope of this chapter, we provide sufficient detail to both understand the measurement methodology and at least to first order, reproduce our experiments.

Inside the operating system kernel, the framework presents two interfaces: one to device drivers, which register with the framework and specify what algorithms and modes of operations they support; and one to applications (e.g., IPsec or */dev/crypto*), which create "sessions." Sessions create context in specific driver instances selected by the framework based on a best-match basis with respect to the algorithms used. Applications queue requests on sessions, and the cryptographic framework, running as a kernel thread and periodically processing all requests, routes them to the appropriate driver. Once the request has been processed, a callback function provided by the application is invoked that continues processing. A software pseudo-driver registers with the framework as a default when no

Copyright © 2005, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

hardware acceleration is available. Public key operations are modeled in the same way, although no session is created.

In summary, the framework provides asynchronous operation, load balancing, application and cryptographic provider independence, and support for both symmetric and public key operations. For our discussion, the most important attribute of the framework is that it provides an *identical common path* to the cryptographic providers available in the system, regardless of their nature (hardware vs. software) or other characteristics (performance, details of the card interface, etc.).

The framework is implemented and has been in use with IPsec since OpenBSD 2.8, although it continues to evolve in response to new requirements. Public key support and the */dev/crypto* API were introduced in subsequent versions of OpenBSD. The OpenSSL crypto library uses this API by default since OpenBSD version 3.1. The OCF has also been ported to FreeBSD, and we are working on Windows and Linux versions.

Extended Attribute File System

The Extended Attributes work from TrustedBSD is an extension to the BSD UFS layer that allows new meta-data to be persistently associated with filesystem objects (files and directories). These meta-data are arbitrary *(name, value)* pairs and can be used to implement Access Control Lists, Sensitivity Labels, POSIX process capabilities, SubOS user IDs (Ioannidis, Bellovin, & Smith, 2002), etc. Besides the obvious extensions to UFS, there are API modifications to accommodate handling of Extended Attributes, as well as the necessary userland tools to manage them.

This work was introduced for TrustedBSD (Watson, 2000), but given the similarity of the kernels, it was believed to be fairly straightforward to import it to OpenBSD and integrate it with the rest of our security architecture. The combination of the /dev/policy interface (Ioannidis, Keromytis, Bellovin, & Smith, 2000), Security-enhanced Linux features, and Extended Attributes should result in a very flexible security enforcement mechanism.

The enhanced file system has been designed and implemented by author Dale Rahn in concert with Robert Watson of NAI Labs/TrustedBSD. The implementation effort has been kept completely synchronized with that of TrustedBSD.

The /dev/policy policy device has been implemented for OpenBSD and continues to be refined. As a major goal of this work was support for SE-Linux, we also undertook an effort (by Tom Langan of Penn) to provide SE-Linux features. For example, the extensions included checking permission on every I/O system call related to files, networks, etc. Conventional BSD systems check just once on `open` or equivalent. This extension was successful and is available, but not in the OpenBSD release. The /dev/policy notions, including the use of advanced policy specification languages, were applied directly.

Copyright © 2005, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

OpenSSL Audit

OpenSSL is used as a technical building block of the secure Apache (Laurie & Laurie, 1999) web server. Web servers are, with considerable accuracy, considered the operating systems of the WWW. Apache is the dominant web server, widely used by commercial and industry sites, and has a greater than 70% market share. Apache provides an operating environment for concurrent transaction processing, script execution, and any other requests that arrive on an HTTP (80) or HTTPS (443) port. The server keeps multiple threads running concurrently to overcome disk and other latencies and provide high performance. A number of services are provided, such as perl scripting, that can help process client PUTs and GETs. When secure Apache is used, the SSL protocol ensures that the transactions with the server are authenticated and encrypted; this behavior is selected, for sites which support it, by prefacing the site name with https: to indicate that the security features are to be used.

The OpenSSL Project is a collaborative effort to develop a robust, commercial-grade, full-featured, and open source toolkit implementing the *Secure Sockets Layer (SSL v2/v3)* and *Transport Layer Security (TLS v1)* protocols as well as a full-strength general purpose cryptography library. The project is managed by a worldwide community of volunteers that uses the Internet to communicate, plan, and develop the OpenSSL toolkit and its related documentation.

A major research issue addressed by POSSE was the portability of the effective OpenBSD audit methodology to other open source efforts. As an experiment, applying the audit methodology to OpenSSL seemed appropriate, given the importance of the OpenSSL software and Apache to electronic commerce. OpenSSL had never been audited, had accreted code from many programmers, and had many patches, and thus was an ideal candidate for the careful scrutiny of a code audit. The strategy we chose was to start the audit with tools, to see what “low-hanging fruit” could be picked by these tools and eliminated in the code base. For example, John Viega’s RATS tool (Viega & McGraw, 2001) can help with fixed-size buffers and detected over 500 instances of fixed-size buffers (which can be exploited for buffer overflow attacks). After some poor initial experiences with RATS, we found that creating search patterns was reasonably powerful. While we looked at Splint (Larochelle & Evans, 2001), we did not end up using it. We were able to detect some errors using a tool supplied by David Wagner (Wagner, Foster, Brewer, & Aiken, 2000).

An important observation about the OpenSSL auditing process is that publicized holes in other systems (for example, on security mailing lists) suggested analogous code in OpenSSL to check, and a variety of problems were identified in this fashion. This suggests that *experience* can play a large role in code auditing, since problematic code will often follow a pattern, which can both be exploited by an experienced attacker and repaired by an experienced auditor. The conclusion at this stage is that tools are an effective way of both pruning low-hanging fruit and identifying chunks of code that need attention. However, many problems still require insight and experience in the auditor.

Copyright © 2005, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

The OpenSSL audit discovered and fixed holes in OpenSSL identified on the Internet. The holes in OpenSSL were fixed just before Defcon and were totally due to CHATS funding. A patch of over 3,000 lines of code secures a host of problems of lesser severity and generally hardens OpenSSL against future attacks. The OpenSSL audit was largely performed by Ben Laurie of AL Digital, Ltd. AL Digital's auditing efforts proved prescient; the fixes in OpenSSL illustrated a potential hole in other systems that was exploited to write the Sapphire/Slammer worm.[2] The worm exploited people who had failed to patch a persistent problem with available security updates (Arbaugh, Fithen, & McHugh, 2000).

A large body of auditing notes and an outline of a book on the OpenSSL audit have been produced, but it is unclear what the final disposition of this information will be. Our operating assumption is that some cleanup and publication on the WWW would extract the maximum value from these unique notes describing both the use of audit tools (such as John Viega's RATS) as well as the manual audit.

Discussion

Almost 37,000 lines of new code are directly attributable to this project (as measured by a scan of the OpenBSD 3.2 source tree), and the POSSE project has directly contributed to the 3.0, 3.1, 3.2 and 3.3 releases of OpenBSD.

In addition, a variety of creative new work has been done. An example of this is the W^X (for Write XOR eXecute) project. This goal of this project is to modify the executable and shared library layout so that the a typical program had no regions of memory that were *both* writable and executable.

This change prevents one of the common attacks where a buffer overflow is used to write code into the address space of a program, then execute that code. This change was introduced with OpenBSD 3.3 on several architectures: alpha, sparc, sparc64. Changes are in progress to add support for this protection to i386 and macppc (PowerPC) architectures with OpenBSD 3.4.

In addition, a modification to GCC called ProPolice written by Etoh was integrated. ProPolice rewrites the layout of stack allocated data including a logical "canary" to detect buffer overruns. This change, coupled with W^X mappings and a randomized stack gap, greatly reduces the chance of a buffer overrun attack being successful.

CONCLUSION

The freedom of open source development has led to a plethora of UNIX-derived and UNIX-like source trees. Each tree has, at best, partially instantiated security features, although OpenBSD has the advantage of audited code. Inadequate resources, insufficient motivation for portable solutions, and too few security experts for all trees have been major barriers. POSSE helps to surmount these barriers and more closely match the resources to the requirements.

Copyright © 2005, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

We have created a project that has been having a substantial impact on the open source community, and beneficiaries have included a variety of commercial vendors who examine or incorporate features from open source systems directly in their own systems. For example, many security appliance vendors use OpenBSD or a minimized version of OpenBSD as the platform for their systems. OpenSSH is shipped with Apple's machines and is extremely widely used.

This has proven a challenging project to manage. There are distributed developers, many distractions, and strong personalities. Nonetheless, we continue to believe that the University is an ideal model for a management entity for this type of effort. By design, its "loose coupling" and open style of discourse provide an easy means by which the long-term goals addressed in the CHATS program can be effectively addressed. Producing a new generation of security-conscious (and capable) developers is a natural enterprise for a University.

ACKNOWLEDGMENTS

This work was supported by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-01-2-0537. Statements made herein are neither explicit nor implied positions of the U.S. Government.

The authors thank Theo de Raadt, the founder and leader of the OpenBSD Project, for his persistence and technical vision.

REFERENCES

Arbaugh, W. A., Fithen, W. L., & McHugh, J. (2000). Windows of vulnerability: A case study analysis. *IEEE Computer*, 33(12): 52-59.

Bach, M. J. (1986). *The design of the UNIX operating system*. Englewood Cliffs, NJ: Prentice Hall.

Brooks, F. P. (1975). *The mythical man-month*. Reading, MA: Addison-Wesley.

Clark, D. D. (1988). The design philosophy of the DARPA Internet protocols. In *Proceedings of SIGCOMM 1988*, 106-114.

Cusumano, M. A. & Selby, R. W. (1997). How Microsoft builds software. *Communications of the ACM*, 40(6): 53-61.

Daley, R. C. & Dennis, J. B. (1968). Virtual memory processes and sharing in MULTICS. *Communications of the ACM*, (5): 306-312.

Fano, R. M. & David, E. E. (1965). On the social implications of accessible computing. In *AFIPS Conference Proceedings 27*, 243-247.

Ioannidis, S., Keromytis, A., Bellovin, S., & Smith, J. (2000). Implementing a distributed firewall. In *Proceedings of Computer and Communications Security (CCS) 2000*, 190-199.

Copyright © 2005, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Ioannidis, S., Bellovin, S., & Smith, J. M. (2002). Sub-operating systems: A new approach to application security. In *Proceedings of the 10th SIGOPS European Workshop*.

Keromytis, A. D., Ioannidis, J., & Smith, J. M. (1997). Implementing IPsec. In *Proceedings of Global Internet (GlobeCom) '97*, 1948-1952.

Keromytis, A., Wright, J., & de Raadt, T. (2003). The design of the OpenBSD cryptographic framework. In *Proceedings of the USENIX Conference*.

Larochelle, D. & Evans, D. (2001). Statically detecting likely buffer overflow vulnerabilities. In *Proceedings of the 2001 USENIX Security Symposium*.

Laurie, B. & Laurie, P. (1999). *Apache: The definitive guide*. Sebastopol, CA: O'Reilly.

Lions, J. (1977a). *A commentary on the UNIX operating system*. Bell Laboratories.

Lions, J. (1977b). *UNIX operating system source code, Level Six*. Bell Laboratories.

McKusick, M. K., Bostic, K., Karels, M. J., & Quarterman, J. S. (1996). *The design and implementation of the 4.4 BSD operating system*. Reading, MA: Addison-Wesley.

Miltchev, S., Ioannidis, S., & Keromytis, A. (2002). A study of the relative costs of network security protocols. In *Proceedings of USENIX Annual Technical Conference (Freenix track)*, 41-48.

MITRE (2003). *Use of Free and Open-Source Software (FOSS) in the U.S. Department of Defense*. MITRE Report MP 02 W0000101, Version 1.2.04.

Organick, E. I. (1972). *The MULTICS system*. Cambridge, MA: MIT Press.

Raymond, E. S. (1999). *The cathedral and the bazaar: Musings on Linux and open source by an accidental revolutionary*. Sebastopol, CA: O'Reilly and Associates.

Ritchie, D. & Thompson, K. (1974). The UNIX operating system. *Communications of the ACM*, 17: 365-375.

Ritchie, D. M. &Thompson, K. L. (1978). The UNIX Time-Sharing System. *The Bell System Technical Journal*, 57(6): 1905-1930.

Saltzer, J. H., Reed, D. P., & Clark, D. D. (1984). End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4): 277-288.

Schroeder, M. D. (1975). Engineering a security kernel for MULTICS. In *Proceedings of the 5th ACM SOSP*, 125-132.

Schroder, M. D., Clark, D. D., & Saltzer, J. H. (1977). The MULTICS kernel design project. In *Proceedings of the 6th ACM SOSP*, 43-56.

Sullivan, G. R. & Dubik, J. M. (1994). War in the information age.

Thompson, K. (1978). UNIX Implementation. *The Bell System Technical Journal*, 57(6): 1931–1946.

Viega, J. & McGraw, G. (2001). *Building secure software*. Reading, MA: Addison-Wesley.

Copyright © 2005, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Wagner, D., Foster, J. S., Brewer, E. A., & Aiken, A. (2000). A first step towards automated detection of buffer overrun vulnerabilities. In *Proceedings of the Symposium on Network and Distributed Systems Security*, 3-17.

Watson, R. (2000). Introducing supporting infrastructure for trusted operating system support in FreeBSD. In the *Proceedings of BSDCon 2000*.

Weinberg, G. (1974). *The psychology of computer programming*. New York: Van Nostrand.

ENDNOTES

[1] See http: //www.cis.upenn.edu/~dsl/POSSE/

[2] See http: //www.cs.berkeley.edu/~nweaver/sapphire/

Copyright © 2005, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

SECTION VI:

IMPLICATIONS OF THE F/OSS DEVELOPMENT MODEL - "THE BROAD PICTURE"

Copyright © 2005, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.