TWiki . CSAC . GuidelinesAndBestPractices

July 2006 Final Draft

Recommendations and Best Practices

There is a continuum from UCAR policy (most formal) to recommendations to best practices (least formal).

- Policies must be adhered to and are not optional. UCAR policies apply to all employees, for example, Access to and Use of Computer and Information Systems. CSAC policies often include enforcement and penalty guidelines.
- Recommendations are important considerations that sysadmins and developers are strongly advised to follow
- Best practices are the larger body of practices that sysadmin and software engineering professionals draw from in creating world-class systems and software

WAG Recommendations

Migrate unnecessary standalone web hosts to a centralized web server

Since it is possible to host numerous virtual hosts on one web server, unnecessary standalone web hosts should be consolidated onto centralized web servers. This will help reduce the number of hosts that need

to be securely configured, patched and monitored, and help consolidate UCAR web applications onto fewer platforms that can be better maintained, patched and upgraded. The WAG recommends that system administrators consider migrating such hosts to UCAR's centralized web hosting service provided by the Web Engineering Group (WEG).

WEG Proxy Service

For divisions, programs and groups that decide to maintain their own web servers rather than utilize the centralized web hosting services provided by the WEG, the WAG recommends placing the web server on an internal subnet and utilizing the WEG Proxy Service to proxy requests from outside the UCAR network to the server. This provides the added security of being behind the UCAR firewall while still allowing external traffic to reach the server. Note that the WHIG's PolicyRecommendationsForUCARCSACDivisions will require security hardening of internal web servers since hackers can still exploit CGIs and essentially compromise an internal host.

Apache CGI Execution

The WAG recommends that Apache be configured to only execute CGIs located in a centralized cgi-bin directory. Permissions for this directory should be set to only allow sysadmins to create new files. That way, sysadmins will know about any new CGIs that developers need installed.

example from httpd.conf:

```
ScriptAlias /cgi-bin "path from root to cgi-bin"
<Directory "path from root to cgi-bin">
   AllowOverride none
   Options none
   Order Allow,Deny
   Allow From All
</Directory>
```

Note: this recommendation also applies to multi-file web applications such as those written in PHP and Python that need to be organized into numerous directories. Each directory that contains files to be executed should be identified and enabled, with appropropriate UNIX permissions and web server directives.

Run Apache as an unprivileged user such as "apache" or "webserv"

Run Apache as unprivileged user so any malicious processes spawned by Apache will have less access to system files. Don't run Apache as 'root', or a regular user. "nobody" has traditionally been used by Apache; the "nobody" user originally as was used to map the "root" account over NFS, and hence may still have some privileges in some systems. The Apache user should be set up with the minimal permissions needed to read the web files it will host.

Apache Modules

Apache can run modules that extend it's functionality. The WAG recommends that sysadmins only run modules that are required for a given website. Apache distributions that come with operating systems often have numerous modules already included. It is more advisable to install the base Apache and add modules only as needed. There is no significant security difference between dynamic and static/compiled modules.

Server Certificates

For internal-user web sites that accept HTTPS (SSL) connections for security reasons, the WAG recommends that sysadmins request the Web Engineering Group (WEG) sign their server certificate. This ensures that your site certificate will work with the UCAR Certificate Authority (CA) which all staff and collaborators are encouraged to install. The benefit of this approach is that users will receive fewer alerts when requesting HTTPS URLs. See the instructions for generating a server certificate and asking the WEG to sign it.

Public sites employing HTTPS should obtain certificates signed by well-known Certificate Authorities (CAs). Commercial CA-signed certificates are available for less than $200 per year. Doing otherwise affects site usability, and habituates users to disregard certificate warnings.

Apache SSLCipherSuite Directive

The WAG recommends setting the SSLCipherSuite directive in Apache's httpd.conf to HIGH:MEDIUM. This directive instructs Apache to only accept HTTPS (SSL-encrypted) requests from modern browsers which support Triple-DES and 128-bit encryption.

example: SSLCipherSuite HIGH:MEDIUM

Adopt the OWASP Top Ten Standard

The Open Web Application Security Project (OWASP) Top Ten is a minimum standard for web security that has been adopted by numerous public and private organizations. The WAG recommends that all UCAR web engineers and web developers adopt this standard and ensure that web applications and web pages that use programming languages such as JavaScript, ASP, .Net, ColdFusion, JSP, Java, PHP, Python, PERL, C, C++, etc. adhere to this standard.

Note that PHP-based scripts and applications are particularly prone to attack. Third party and open source applications based on PHP should be agressively kept up to date with patches. PHP applications that are widely and constantly exploited should be avoided (eg. PHPbb). PHP web developers working on in-house development projects should take special care to adhere to the OWASP Top Ten standard.

Adopt the Center for Internet Security's "Apache Benchmark"

The Center for Internet Security (http://cisecurity.org) has published a set of community-developed security benchmarks that allow administrators to quickly assess how closely their systems follow accepted best practices. The Apache Benchmark (http://www.cisecurity.org/tools2/apache/CIS_Apache_Benchmark_v1.0.pdf, registration required) includes a guide to best practices and scoring tools.

Attend annual configuration review meetings

The starting point for assessing the security of a website is reviewing its HTTP server daemon configuration since it contains directives pointing to all of the hosted applications. A configuration review helps pinpoint and correct configuration mistakes, assess access controls, inventory applications, and comment on security status. A thorough review may even reveal previously unknown operating applications. The review can also provide an opportunity to consider additional solutions, such as mod_security, which can improve security and enhance logging.

Configuration review at UCAR may encounter institutional or cultural resistance. Bear in mind that it is less of a burden than complete code review, and will yield improvements to our web security stance.

To make the review less of a burden, the WAG will sponsor annual configuration review "parties" to

support this work. Having a second person review a configuration will help locate errors, provide input, and clarify poorly-documented sections. Doing this as a social function will make the process less onerous and improve compliance. For the sake of consistency, this should be held on the same month every year, say, October.

Use the UCAS authentication system for web applications

Web developers are more likely to get authentication right if they use a locally-supported system, and users are more likely to use strong passwords if they have fewer to remember. The Web Engineering Group and UCAR security administrators will provide libraries, examples and documentation on using the UCAS passwords for web authentication.

Best Practices

N-Tier Server Architecture

For web servers that sit on an exposed subnet outside the UCAR network security perimeter, it is advisable to use an n-tier architecture, where the exposed server runs a minimum of services (i.e. Apache and/or Tomcat) and another server provides other required applications such as databases and special purpose applications. This prevents the exposed server from being compromised by vulnerabilities in databases and special purpose apps. The idea here is to harden the front-end server as much as possible and have it communicate with a back-end server inside the security perimeter for additional required services.

Security in Depth: System Administration Security Best Practices

There are numerous layers of security, each of which should be addressed for a server providing web applications:

- Network (block unnecessary ports with a firewall, subnet filter, or ipfilter)
- Operating System (harden, turn off unnecessary services)
- File System (lock down permissions, design a secure file hierarchy)
- Applications (keep patched, use best practices in configuring, review CGIs)

This is of course a huge topic. Important top-level considerations covered in a white paper published on the SAN site by Harish Setty include:

- Keep your knowledge up to date (read security bulletins for your OS and apps you host, keep abreast of changes sysadmin peers make on your servers, stay current by attending professional conferences and reading publications)
- Secure the physical system and console
- Harden the system (run the minimum services and packages necessary, disable all others, close unnecessary ports and use TCP wrappers to restrict incoming connections)
- Keep services and packages patched and up to date (especially Sendmail, BIND and PHP web applications which are common targets)
- Superuser password practices (require One-Time Passwords such as CryptoCard? for sysadmin access, require sudo for superuser access to provide an audit trail)
- User education (educate any system users to use secure passwords and protocols)
- Vulnerability Testing (use security audit tools to scan your systems, think like a hacker and try to find vulnerabilities you have left open)
- Monitor systems (use logging and monitoring software to keep an eye on what's happening on your systems)

- Intrusion detection (use a host-based intrusion detection solution like Tripwire to detect unauthorized server modifications, this makes recovery much more manageable)
- Backup and disaster recovery (develop a solid plan for recovering from crashes, failures, and intrusions, keep backups up to date)

Please see the ConfigurationChecklist for a list of more specific considerations.

Additional security best practice resources are available from:

- Sun Microsystems

Consult with your system administrator, CSAC representative, or the UCAR computer security team for guidance on securing your web server to the appropriate standards.

Development Process

Software engineering projects greatly benefit from a rational development process. While there are many to choose from (Extreme, Agile, Waterfall, etc.), all share some common phases.

- Gather requirements by interviewing stakeholders and users
- Create a requirements document
- Review and refine requirements
- Create a design document
- Develop
- Team code reviews
- Test
- Launch

Every web application project should include a security section in its requirements document. By including security as a consideration from the start, you will avoid having to make code modifications for security in the middle of coding or worse yet after coding is finished. Such changes are disproportionately expensive and can lead to far weaker "bolted on" security approaches.

Also include the testing of security requirements in your test plan.

Developing Secure Code

Open Web Application Security Project (OWASP) Top Ten

OWASP surveyed a variety of security experts around the world to identify a broad consensus about what the top ten web application security flaws are. It represents a minimum standard for web security that has been adopted by numerous public and private organizations. As of Sept 2005, the list was:

1. Unvalidated Input
2. Broken Access Control
3. Broken Authentication and Session Management
4. Cross Site Scripting (XSS) Flaws
5. Buffer Overflows
6. Injection Flaws
7. Improper Error Handling
8. Insecure Storage
9. Denial of Service
10. Insecure Configuration Management

Please see the OWASP Top Ten website for more information and specifics for how to deal with these vulnerabilities. This document should be required reading for all web engineers and web developers.

Build Security In

Build Security In is a resource for designing and developing more secure systems. The site has patterns that developers and architects can use to address security from the earliest stages of development.

WebGoat

For an excellent introduction to secure coding, try WebGoat, an interactive teaching environment for learning about web security. In each lesson, users must demonstrate their understanding by exploiting a real vulnerability on the local system. Please do not install this J2EE web application on production systems. Use it on your own local desktop computer.

Please see the EducationPlan and CodeExamplesCodeLibrary pages for a list of secure coding best practices, examples, and book references.

Develop in a Framework that Supports Secure Coding

Web application frameworks provide a level of abstraction between application functionality and low-level details like page navigation or file and database access. Good frameworks can enhance security by providing quality libraries for interacting with the operating system or the back-end database, and insulate programmers from lame mistakes. Model-view-controller (MVC) frameworks are especially popular, and examples in include RubyOnRails, Django (Python), Spring (Java/J2EE), Cake (PHP).

ACEGI

ACEGI is an authentication and authorization system for Spring, a Java framework. It has ready built servlet filters that provide extensive control and in many cases just require configuration rather than coding.

Self-audits and External Audits

Designing systems and software with security in mind is an important step, but it is just as important to verify success with self-audits, and where sensitive data is involved, consider audits by an external firm. There is no substitute for trying to hack your own code. It gets you thinking about potential vulnerabilities and can identify them before hackers do. Some important strategies include:

- review open ports and services running and turn off any that aren't required (Nessus and Nikto
- review file permissions (check for world writable files and lock down cgi permissions)
- attempt intrusions and hacks, test your own code by submitting shell commands in form input, submit buffer overflows, SQL injections, etc.
- Search Google for pages on your site that should be secure to make sure they are not found. If they are found, see Google's instructions on removing indexed and cached pages.

Security Assessment Tools

While web security assessment tools are not a solution to security problems, they can suggest potential vulnerabilities. They include:

- Nessus - Security scanner for Linux, BSD, Solaris, and other flavors of Unix that performs over 900 remote security checks, and suggests solutions for security problems
- Nikto - Web server scanner which performs comprehensive tests against web servers for multiple items, including over 3200 potentially dangerous files/CGIs, versions on over 625 servers, and version specific problems on over 230 servers
- WebScarab - a framework for analysing web applications, it records the conversations (HTTP and HTTPS requests and responses) that it observes, and allows the operator to review them in various ways for debugging or vulnerability analysis
- Whisker - a CGI scanner written in PERL

Web Application Firewalls (WAF)

Web Application Firewalls (WAF) are an emerging security technology that prevent attacks on web applications that cannot be caught by network firewalls and intrusion detection systems. It is not necessary to modify application code.

The Web Application Security Consortium (WASC) has published a set of evaluation criteria for WAF products.

mod_security is an open-source WAF that runs within Apache, and can provide extensive filtering and logging either as a running module or when deployed as a reverse proxy. UCAR groups should investigate mod_security as a defensive measure for applications pending their code review and refactoring, and perhaps indefinitely. A typical configuration would deploy mod_security to filter SQL injection, filter cross-site scripting, normalize URLs (e.g., convert "path/../../something" to "/something"), rewrite input to white listed characters, and more. Sufficient interest within UCAR may suggest that the WEG deploy a central WAF as a reverse proxy for division and group servers.

----- Revision r1.18 - 06 Jul 2006 - 18:11 GMT - PeterBurkholder