Applied Web and Network Security

Achim D. Brucker
brucker@inf.ethz.ch
http://www.brucker.ch/

Information Security
ETH Zürich
Zürich, Switzerland

Information Security Fundamentals
March 23, 2004

Motivation

HTTP in a Nutshell

The End Users View

The Server Providers View

Conclusion

Motivation

Over 90% of online apps not secured against common cracking techniques!

A research of WebCohort's Application Defense Center revealed the most common vulnerabilities for web applications in 2003:

- Cross-site scripting (80%).
- SQL injection (62%).
- Parameter tampering (60%).
- Cookie poisoning (37%).
- Database server (33%).
- Web server (23%).
- Buffer overflow (19%).

What is Web Security?

- Web security is not as well-defined as e.g. cryptographic security.
- Practical web and network security depends on
  - details of network standards,
  - implementation details,
  - concrete versions of browsers and servers.
  - ...
- Attacks against privacy, security, and quality of service ("safety").
- Web and network security is a "moving target".
- There is no "once and forever" solution.

Roadmap

Motivation

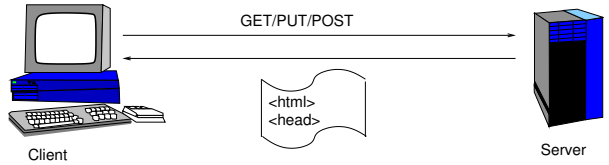
HTTP in a Nutshell

The End Users View

The Server Providers View

Conclusion

HTTP in a Nutshell



- HyperText Transfer Protocol (HTTP) is defined in RFC 2068.
- HTTP is an application level protocol.
- HTTP transfers hypertext requests and information between server and browsers.

HTTP: The Client Side

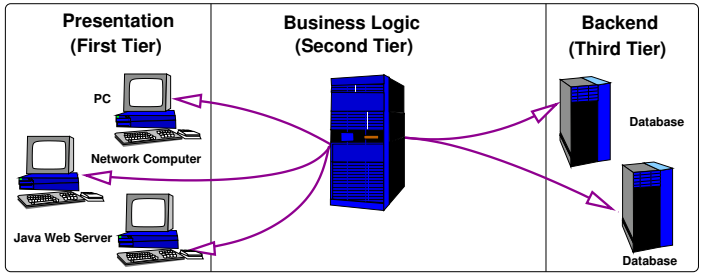
- The client initiates all communication:

Method	Description
GET	request a web page
HEAD	request header of a web page
PUT	store a web page
POST	append to a web page

- The user navigates trough URLs, e.g. http://www.infsec.ethz.ch/.
- HTTP does not support for sessions.

HTTP: The Server Side

- The server delivers data upon request of the client.
- Arbitrary data can be transfered (client takes care of processing).
- The data can be computed on demand (web application) or can be static (HTML pages, images, . . .).
- Three tier architecture is widely used:



Roadmap

Motivation

HTTP in a Nutshell

The End Users View

The Server Providers View

Conclusion

HTTP Header

- On each request, the client sends a HTTP header to the server.
- Normally headers are sent unencrypted.
- Headers contain information such as
  - requested language,
  - requested character encoding,
  - used browser (and operating system),
  - ...
- HTTPS sends headers encrypted.

HTTP Headers: Private Information

- HTTP headers can also contain "private" information, e.g.:
  - FROM: the users email address, critical due to user tracking and address harvesting (spam).
  - AUTHORIZATION: contains *authentication* information.
  - COOKIE: a piece of data given to the client by the server, and returned by the client to the server in subsequent requests.
  - REFERER: the page from which the client came, including search terms used in search engines.
- Combining information (e.g. FROM, REFERER, IP address) allows server providers already a reasonable tracking of the users behavior.
- **Remark:** in HTTP, "authorization" *means* "authentication"!

Cookies

- Cookies were introduced to allow session management.
- The main idea is quite simple:
  - A server may, in any response, include a cookie.
  - A client sends in every request the cookie back to the server.
  - A cookie can contain any data (up to 4Kb).
  - A cookie has a specified lifetime.
- Cookies received lots of criticism for privacy reasons.

Cookies and Privacy

- Cookies can be used to track users.
- Privacy is attacked from many sides:
  - Analyzing server logs.
  - Eavesdropping traffic (even encrypted headers are informative).
  - Enforcing proxys (or application level firewalls), e.g. deployed by your ISP or employer.
  - Reveal "browser logs" (e.g. history) on the client side.
- Thus, cookies are only part of the game.
- Anyway, cookies should be considered as confidential information!
- Cookies with very long lifetimes are suspicious!

HTTP: Authentication

HTTP supports two authentication modes:

- **Basic authentication:**
  - Login/password based.
  - Information is sent unencrypted.
  - Credentials are sent on every request to the same realm.
  - Supported by nearly all server/clients and thus widely used!
- **Digest authentication:**
  - Server sends nonce.
  - Client hashes nonce based on login/password.
  - Client sends only cryptographic hash over the net.
  - Seldom used.
- Use browser features for storing your login/password with care!

General Considerations

- Be careful when using public web browsers (e.g. internet cafe).
- Visited sites are stored
  - in the browsers history,
  - in the browsers cache,
  - can also be revealed by auto-completion features.
- Use the "manage password" feature with care.
- Many threats are caused by malicious active components (JavaScript, ActiveX, . . .).
- Browsing the web is not as harmless as it should be!

Roadmap

Motivation

HTTP in a Nutshell

The End Users View

The Server Providers View

Conclusion

The Most Critical Web Application Security Vulnerabilities

Software is generally created with functionallity at frist in mind and with security as a distant second or third.

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.

What have these threats in common?

- They attack neither cryptography nor authorization directly.
- They all exploit programming or configuration flaws.
- All of them are relatively easy to exploit.
- They all can cause serious harm,
  - either by revealing secret data,
  - or by attacking quality of service.
- They can only be prevented by well-designed systems.

Unvalidated Input 1/2

- Note:
  - Web applications use input from HTTP requests.
  - Attackers can tamper any part of a HTTP request.
- Main idea: send unexpected data (content or amount).
- Possible attacks include:
  - System command insertion.
  - Cross-site scripting.
  - Exploiting buffer overflows.
  - Format string attacks.
  - SQL injection.
  - Cookies poisoning.
  - Manipulating (hidden) form fields.

Unvalidated Input 2/2

- Many sites rely on client-side input validation (e.g. JavaScript).
- Ways to protect yourself:
  validate input against a positive specification.
  - Allowed character sets.
  - Minimum and maximum length.
  - Numeric ranges.
  - Specific patterns.
- Only server side input validation can prevent these attacks.
- Applications firewalls can provide only some parameter validation.
- These kind of attacks are becoming more likely!

Broken Access Control

- Reliable access control mechanisms are
  - difficult to implement.
  - difficult to configure, setup and maintain.
- Access control policy should be clearly documented.
- Rethink your requirements and scan your setup for:
  - Insecure IDs: is an attacker able to guess valid IDs?
  - Forced browsing past access control checks: can a user simply access the protected area directly?
  - Path traversal: take care of absolute and relative path names.
  - File permissions.
  - Client side caching.

Broken Authentication and Session Management 1/2

- Authentication and session management includes web pages for
  - changing passwords.
  - handling of forgotten passwords.
  - updating (personal) account data.
- The complexity of such systems is often underestimated.
- An attacker can hijack a user's session and identity.

Broken Authentication and Session Management 2/2

To avoid these treats a web application should:

- Require to enter the login password on every management site.
- Require strong passwords.
- Implement a password change control.
- Store passwords as hash (whenever possible).
- Protect credentials and session ID in transit.
- Avoid browser caching.

Why not switch to HTTPS (SSL)?

Cross-Site Scripting (XSS) 1/2

- The attacker tries to inject malicious code in well-known sites.
  $\Longrightarrow$ Users will trust this code!
- Assume we access http://www.abcd.com/mypage.asp and get:

  Sorry http://www.abcd.com/mypage.asp does not exist
- what happens, if we replace "mypage.asp" with a malicious script?
- we get a page from a trusted site (www.abcd.com) with malicious content,e.g:
  http://www.abcd.com/<script>alert(document.cookie);</script>
  can be used to steel cookies!

Cross-Site Scripting (XSS) 2/2

- For example, we could mail this error page to our victim.
- Our victim's browser will execute the script (from a trusted site).
- More easy: copy malicious content into trusted message boards.
- XSS can be used to steal session IDs of valid users.
- XSS is a special form of unvalidated input attack.

Buffer Overflows

- Buffer overflows are caused by "sending too much data".
- Buffer overflows corrupt the execution stack of the application.
- Buffer overflows can occur in any software
  worthy exception: languages with runtime checking, e.g. Java.
- To prevent buffer overflow attacks:
  - watch for bug reports and install patches timely.
  - program your own applications "for safety"!
- Overflow attacks are common for operating system attacks

Injection Flaws

- A special injection "unvalidated input" attack.
- Attacker tries to inject commands to the back-end system.
- Back-end systems include:
  - the underlying operating system (system commands).
  - the database servers (SQL commands).
  - used scripting languages (e.g. Perl, Python).
- The attacker tries to execute program code on the server system!

Injection Flaws: SQL Injection

- Assume a web application with a database back-end using:
  SELECT * FROM users WHERE user='*$usr*' AND passwd='*$pwd*'
- What happens if we "choose" the following value for *$pwd*:
  ' or '1' = '1
- We get
  SELECT * FROM users WHERE user='*$usr*' AND passwd='' **or '1' = '1**'
- As '1' = '1' is valid, we will be authenticated!

Preventing Injection Flaws

- Filter inputs (using a list of allowed inputs!).
- Avoid calling external interpreters.
- Choose safe calls to external systems.
- For databases: prefer precomputed SQL statements.
- Check the return codes to detect attacks!

Improper Error Handling

- Error messages reveal details about your application, especially if they contain stack traces, etc.
- Do not distinguish between “file not found” and “access denied”.
- Your system should respond with short, clear error messages to the user.
- Execution failures could be a valuable input to the intrusion detection system.

Insecure Storage

Using insecure storage can have many reasons:

- Storing critical data unencrypted.
- Insecure storage of keys, certificates.
- Improper storage of secretes in memory.
- Poor choice of cryptographic algorithms.
- Poor sources of randomness.
- Attempts to invent “new” cryptography.
- No possibility to change keys during lifetime.

Preventing Insecure Storage

To prevent insecure storage:

- Minimize the use of encryption (“it’s secure, it’s encrypted”).
- Minimize the amount of stored data (e.g. hash instead of encrypt).
- Choose well-known, reliable cryptographic implementations.
- Ensure that keys, certificates and password are stored securely.
- Split the master secret into pieces and built it only when needed.

Denial of Service

- Beside network (e.g. SYN floods) also application level DoS.
- In principle: send as many HTTP requests you can.
- Today: tools for DDoS available for everyone.
- Test your application under high load.
- Load balancing could help.
- Restrict number of requests per host/user/session.

Insecure Configuration Management

Maintaining software is a difficult problem and not web application specific. You should

- never run “unpatched” software.
- carefully look for server misconfigurations.
- remove all default accounts with default passwords.
- check the default configuration for pitfalls.
- remove unnecessary (default) files (e.g. default certificates).
- check for improper file and directory permissions.
- check for misconfiguration of SSL certificates.

Roadmap

Motivation

HTTP in a Nutshell

The End Users View

The Server Providers View

Conclusion

Conclusion

- Many security problems in practice are caused by the complexity of systems built, e.g.:
  - by combining small systems into larger ones.
  - by (slightly) incompatible implementations.
  - complex configuration issues.
- Remember: systems are only as secure as the weakest link!
- Today, cryptography is difficult to crack, but (concrete) systems built are vulnerable.
- Most successful attacks build on programming and configuration errors.

Security Guidelines 1/2

- Design:
  - Keep it simple.
  - Security by obscurity won't work.
  - Use least privileges possible.
  - Separate privileges.
- Implementation:
  - Validate input and output of your system.
  - Don't rely on client-side validation.
  - Fail securely (closed).
  - Use and reuse trusted components.
  - Test your system (e.g. using attack tools).

Security Guidelines 2/2

- Additional techniques:
  - You should not rely only on a "standard" firewall (filtering IPs and ports):
    you have to filter carefully on the application level!
  - Application level firewalls can help, but are not an all-in-one solution.
  - Apply intrusion detection.
- Security issues are changing every day: keep up-to-date!
- Review your setup regularly!

Further Reading

- William Stallings, *Cryptography and Network Security*, Prentice Hall, 2003
- The Open Web Application Security Project, `http://www.owasp.org`
- The Ten Most Critical Web Application Security Vulnerabilities, OWASP, 2004, `http://www.owasp.org/documentation/topten`
- A Guide to Building Secure Web Applications: The Open Web Application Security Project, OWASP, 2004, `http://www.owasp.org/documentation/guide`
- David Scott and Richard Sharp, *Developing Secure Web Applications* in IEEE Internet Computing. Vol. 6, no. 6. Nov/Dec 2002. `http://cambridgeweb.cambridge.intel-research.net/`