Toward an Automated Vulnerability Comparison of Open Source IMAP Servers

Chaos Golubitsky (chaos@glassonion.org)

December 7, 2005
Overview

- Motivation
- Attack surfaces
- IMAP server design
- Automated attack surface measurement
- Results
Why automated vulnerability comparison

• Goal is to minimize vulnerability of installed software

• But how do we measure future vulnerability?
  – Past bugs?
  – Reputation?
  – Look at the code?

• Useful metric must:
  – Be easy to apply
  – Give simple and usable results
Why analyse IMAP servers

• Protocol for remote authenticated e-mail access

• Three popular servers: UW, Cyrus, Courier

• Prior vulnerability data is inconclusive:
  – Approximately 30 IMAP server vulnerabilities recorded
  – Almost all are remote API buffer overflows allowing arbitrary code execution
Analysis methodology: attack surfaces

• Methodology for generating metrics

• Two prerequisites for an attack on a system:
  – Before attack, attacker must be able to affect the system in some way
  – Attack must increase attacker’s access to the system

• Measure system attackability by counting ways to affect the system

• What do attack surface elements look like?
The attack surface of an IMAP server
IMAP design choices which affect attackability

- Permissions and authentication:
  - Imapd account (Cyrus) vs. root/user (UW, Courier)

- Subset of functionality which is built in:
  - Needs external network listener (UW)
  - Custom tcpwrappers workalike (Courier)
  - Custom procmail workalike (Cyrus)

- How do we rigorously measure the effects of all these choices?
Measuring attackability: using the source

- Metric used: a weighted count of the code functions available through the IMAP network interface
- Weighting: not all functions are equally accessible
- Access rights:
  - Authorization needed to execute the code
  - Unauthenticated, anonymous, user, administrator
- Privileges:
  - Power the operating system gives to the running code
  - nobody, user, imapd, root
How to automatically count and classify functions

• Use a code analysis tool to find all reachable functions

• Starting from \texttt{main()}, manually divide code by privilege/access

• Finding privilege/access boundaries:
  – Privilege: look for \texttt{setuid()}/\texttt{setgid()} calls
  – Access rights: password checks? internal variables?

• Output: set of functions accessible at each privilege/access level
From sets of functions to an attackability value

- Assign a weight to each privilege and access level:
  - More privileged functions have higher weight:
    * weight(root) > weight(nobody)
  - Functions with more access restrictions have higher weight:
    * weight(authenticated) > weight(unauthenticated)

- Choose a simple attackability function satisfying:
  - Higher privilege leads to higher attackability
  - Higher access restriction leads to lower attackability

\[
\text{Attackability(codebase)} = \sum_{f \in \text{functions}} \frac{\text{weight(priv}(f))}{\text{weight(access}(f))}
\]
Results and Discussion

Courier outperformed others significantly, while UW and Cyrus tied

- Metric rewards privilege separation heavily:
  - Courier designed to have good privilege separation
  - Cyrus contains more code than UW, but scored similarly

- Results can depend on specific numerical weights chosen:
  - Cyrus imapd account vs. UW root/unprivileged user
  - Imapd almost as privileged as root? UW wins
  - Imapd barely more privileged than user? Cyrus wins

- Still needed: better automation, comprehensiveness

- As implemented, attackability metric gives some sensible results
Questions?

Toward an Automated Vulnerability Comparison of Open Source IMAP Servers

Chaos Golubitsky (chaos@glassonion.org)

December 7, 2005