

- Why should you care?
 - Web-based attacks are here
 - 70% of successful attacks through layer 7 (Gartner)
 - Web-based attacks are not going away
 - Low barriers of entry
 - Lax security
 - Vulns are everywhere
 - Vulns are easy to find
 - Web-based attacks are high profile
 - Paris Hilton T-Mobile hack
 - MySpace.com attack
 - Web-based attacks forces VISA PCI CISP
 - Automating these attacks, let alone self-replicating attacks, only make this worse.
- Why do these attacks happen?
 - Web applications are complex
 - Multiple techs spanning multiple disciplines
 - "Not my responsibility"
 - Program design vs. program implementation
 - Existing disconnect between programmers and security
 - Popularity of web apps like Google Maps, Gmail Outlook Web Access, Salesforce.com, and Flickr driving more unaware programmers into the field
- Clearing up some myths
 - Layer 7 is dominated by very simple protocols
 - FTP
 - Telnet
 - SMTP, POP
 - We are concerned with HTTP/HTTPS/WebDav
 - Simple != limited
 - Common Myths
 - SSL
 - SSL is not Layer 7!
 - SSL does not secure an applications
 - SSL secures the communication
 - Everything in presentation works regardless of SSL
 - XSS has limited impact
 - AJAX, iFrame remoting, RegExs make XSS much worse than 6 years ago
 - XSS creation is easy to automate
 - Payload creation can be separated from payload
- Web Worms and Web Viruses

- Traditionally attacks are still plentiful
- 2005 saw release of self-replicating programs that automatically exploit web app vulns.
- Web Worms
 - Conventional and XSS worms
 - Propagate from host to host
 - Language independent
 - Somewhat OS independent
 - Spreads itself by sending requests to run on vulnerable hosts that then run worm
 - Runs on servers
 - Payloads can be pretty much anything
- Web Viruses
 - Infects different pages on same host
 - Written in Javascript
 - Completely OS independent
 - Simply viewing a page with a browser infects new pages
 - Runs inside browser on client
 - Payloads are bad, even with DOM restrictions
 - Keylogging
 - Form scrapping
 - Remote control, arbitrary requests
- How Web Worms and Web Viruses Propagate
 - Both propagate exploiting some vulns in a web app
 - Send a specially crafted request which allows code execution, injects code into into a database of dynamic content, etc.
 - All requests travel over HTTP
 - Difficult to detect and stop layer 7 attack requests
 - Besides port 53, port 80 is most open port on firewalls
 - Web application firewalls are not generally effective against complex web apps
 - Profiling "normal" behavior to detect bad doesn't work
 - "Normal" is constantly changing with types of users, load, (ex: holiday shopping)
 - Normal site use can look like attack
 - Large POSTs (.NET's viewstate)
 - Complex forms/parameters with funny characters
 - Users from all over the globe.
 - Anonymity enhancements breaking state
 - IDS/IPS evasion at layer 7 is easier than other layers
 - Packet-based vs. stream-based IDS/IPS (David

Maynor/Robert Graham

<http://www.blackhat.com/presentations/bh-jp-05/bh-jp-05-maynor-graham.pdf>)

- IP Fragment hanging - IDS/IPS and web server see different requests (Dan Kaminski
<http://www.toorcon.org/2005/slides/dkaminsky-blackops2005.pdf>)
- Many different ways to send attack strings
 - Conventional Web Worm
 - Executing code on server, anything you want
 - Perl::LWP, Sockets, even netcat
 - XSS Worm, Web Virus
 - Restricted by Javascript
 - Unidirectional (browser -> host) AKA blind requests
 - Arbitrary GETs to any domain
 - Image Object
 - Script Object
 - Arbitrary POSTs to any domain
 - Javascript createElement to build FORM
 - document.forms[0].submit
 - Bi-directional (browser <-> host)
 - Arbitrary HTTP to same domain
 - AJAX (example)
 - Arbitrary HTTP to **any domain**
 - Big tangent to this talk. Yes it can be done. It can be used for Very Bad Thing(tm). BlackHat Las Vegas 2006, baby!
 - Web Worms (Detailed)
 - 2 Types
 - Only conventional ones have been "in the wild"
 - Conventional Web Worm
 - Run by underlying OS on server
 - Almost any language: Perl, Python, other interpreted languages allow some OS independence
 - Exploit vulns in web apps that allow remote code execution
 - Buffer overflows
 - SQL Injection
 - CGI Bugs
 - Once spread, executes by
 - Worm already can execute anything on browser
 - Limitations
 - User account that web server or module runs under

- Underlying OS
- XSS Web Worm
 - Theoretical (MySpace.com attack was Web Virus)
 - Run inside the browser on the client
 - Language: Javascript
 - Exploits XSS vulns to get script into other web pages on other hosts
 - XSS vulns are laughably common
 - Once spread, executes by
 - If XSS script got into backend database
 - Website will serve it to users, executes
 - Message boards, forums, user profiles
 - Limitations
 - Few Imposed by Javascript.
 - Reference payloads in XSS-Proxy, Phuture of Phishing.
- Web Virus (Detailed)
 - Has been seen in the wild (MySpace.com attack)
 - Backend databases for dynamic content are injected with XSS code
 - XSS code served with page view, browser executes XSS which infects more pages.
 - Virus? Sure
 - Infects pages/databases on same host
 - Increases exposure of virus, virus runs more often
 - Payloads
 - Geared more towards info stealing than host damage
 - Limitations actually prevent most host damage
 - Implications of a Web Virus
 - Huge! Virus runs in the browser
 - Truly cross platform instead of carrying payloads for multiple platforms
 - Immune to conventional virus detection
 - Virus is stored database with other highly dynamic content
 - Scanner software has file system hooks, not DB hooks
 - Server filesystem, code paths, binaries are never modified
 - How do you even detect malicious Javascript code?
 - To client, came from same origin, the actual server
 - Malicious Javascript looks just like regular Javascript
 - Requests images, possible from other

- domains (images.domain.com)
 - Manipulates/changes DOM tree
 - Hooks OnEvents
- Come on, "huge"? Aren't you just selling fear?
 - Compare traditional info stealing Trojan to a Web Virus
 - Image a Web Virus that uses Javascript to capture keystrokes, send to 3rd party
 - Has infected a user's calendar page on a CRM
 - Any other user who views infected page gets their calendar infected (AJAX, blind POST, etc), spreading the virus
 - Infected pages also has Javascript to log keystrokes
 - Keylogger persists across that whole domain (XSS-Proxy, iframe magic), logging keys on all web pages
 - Tripwire, integrity checks all pass. ISRs intact, no cloaked processes, no IPC, server binaries not modified, user's browser not modified. No trace that keylogger is running!
- Analysis of a Real Web Worm and Web Virus
 - Perl.Sanity
 - Conventional Web Worm
 - January 2005, some variants in spring 2005
 - Perl (walk through code)
 - Attack Vector: Code execution in input
 - Propagation:
 - Google searches with static string to find vuln hosts
 - Perl::LWP to make GET request with attack parameter
 - Payload
 - Page replacement
 - Trivial to fix
 - Analysis
 - Google search provided choke point.
 - Host selection very poor, kept infecting same hosts
 - No mutation of code, search string, attack string
 - Payload was silly
 - MySpace.com attack
 - Web Virus
 - October 2005

- JavaScript with AJAX (walk through code)
- Attack Vector: XSS Exploit allowed <SCRIPT> in profile
- Propagation:
 - Used AJAX to inject virus into MySpace profile of any user viewing infected page.
- Payload
 - Used AJAX to force any user viewing infected page to add user "Samy" to their friend list.
 - Appends the words "Samy is my hero" to victims profile
- Analysis
 - Awesome Hack!
 - No, I didn't write it (Toorcon foreshadowing)
 - Great proof of concept about using that AJAX is a security risk even though it follows DOM security.
 - MySpace lucked out; could have been much worse payload
- Walk thru of infection scenario.

- Truly evil Web Worms and Web Viruses
 - So, what is the worst case with these threats?
 - Swogmoh Web Worm
 - Hypothetical Conventional Web Worm
 - "Holy Mother of GOD, We're Screwed!" backwards
 - Perl::LWP
 - Attack Vector: Multiple SQL Injection vulns of different apps
 - Propagation:
 - Use Google, others to find new sites vuln to one of our SQL exploits
 - Mutate search string
 - Allinurl: ~= inurl:
 - Word order
 - Algorithm to generate nonsense English words
 - Uses code from Nikto web scanner to brute find new SQL Injection vulns
 - More data? maybe
 - More infections? maybe
 - More traffic? tons!
 - Payload Web
 - Known vulns = known app = known database structure
 - Dump usernames/passwords to mailing lists, blog comments
 - Listen to the sound of 100,000 DROP TABLEs

- INSERT INTO databases with garbage, bad ASCII, etc
 - Random Nikto scans are basically a DDoS
 - 1929 Virus
 - Hypothetical Web Virus
 - Targets a major stock trading site (E-Trade, Ameritrade, etc)
 - Javascript + AJAX
 - Attack Vector: XSS exploit to get <SCRIPT> into forum, user profile, stock report, etc
 - Propagation:
 - Use AJAX/blind POST to inject virus into other pages using credentials of any user viewing infected page.
 - Payload:
 - Using AJAX to make buy and sell orders on your behalf
 - MySpace.com had complex confirmation pages too. Not an issue.
 - 2 Modes:
 - Online mode
 - Javascript uses heartbeat, super secret bidirectional tunnel to receive commands from 3rd party
 - Allows massive damage to entire stock market as thousands of users simultaneously sell otherwise healthy stocks
 - Research mode
 - Use AJAX to monitor certain stocks on the site, actively looking at rate of change of the stock to detect buying/selling trends
 - Identify stocks that are starting decline or are bad
 - Buy them just as they start to decline, sell when dirt cheap.
 - Damages a lot of individual portfolios.
 - Imagine 100,000 users trying to convince the SEC they really didn't make a trade.
 - Even if mess does get resolved, buys and sells having changed market and caused other buys and sells by groups external to infected site.
- Preventing Web Worms and Web Viruses
 - Ultimately, Worms and Viruses are exploits web app vulns
 - Fixing vuln will stop both parts of the infection

- Initial injection and further propagation
- Payloads
- Your web applications are the bricks in the castle wall around your website. Do you really trust a brick you downloaded off SourceForge?
- Your inputs are your doors and windows; only allow what you absolutely must
- Guidelines
 - Never trust anything you get from the client. Everything can be modified!
 - "Hidden" HTML Input tags
 - Cookies
 - URL parameters
 - POST Data
 - HTTP Headers (Referer, etc)
 - Never used input you get from the client without sanitizing it
 - Enforce types: Only numbers? Only letters? Formatting?
 - Must be this tall to ride; check the size of the request (TinyDisk file system)
 - Escape characters like < " & ` | and > to avoid XSS and SQL Injection vulns.
 - PHP, ASP, etc have all have built in functions for this
 - Validators on both sides
 - Client-side validators exist **solely** for performance issues
 - Serve-side validators are the only way to enforce the rules
 - Web frontend needs to represent backend code
 - FORM POSTs backend code should only allow access to POST variables (Request vs. Request.QueryString)
 - Do you need a full relational database? Would a LDAP directory work?