Analysis of Web Application Worms and Viruses

Billy Hoffman (bhoffman@spidynamics.com)

SPI Labs Security Researcher
Presentation Outline

• Why you should care
• Why these attacks happen
• Web application worms and viruses
• Analysis of Perl.Santy, MySpace.com, and Yamanner web malware
• Hypothetical, worst case examples of web malware
• Guidelines for writing secure web applications
Why You Should Care
Why You Should Care

- Web applications are not going away
- Offer too many advantages to be ignored by businesses
  - Browser is a ubiquitous platform available on all operating systems and patch levels
  - Central location solves deployment, incompatibilities, and diverse deployed version issues
  - Easy to maintain a single server copy of software
  - Appealing for budgets: cheap to deploy and maintain
  - Large companies adopting web applications
    - Salesforce.com
    - Google’s various apps
    - Microsoft’s “upcoming” Windows Live, Office Live
Why You Should Care

- Web-based attacks are here

“Today over 70% of attacks against a company’s website or web application come at the ‘Application Layer’ not the network or system layer.”

- Gartner Group
Why You Should Care

• Web-based attacks are not going away
  – Low barriers of entry
  – Lax security
  – Vulnerabilities are everywhere
  – Vulnerabilities are easy to find (Long’s *Google Hacking*)
  – Re-use of common components (php[whatever]) makes multiple sites vulnerability to a single issue
  – Even if a site is secure, you have the entire Internet to find other vulnerable sites.
Why You Should Care

- Web-based attacks are high profile
  - Paris Hilton T-Mobile hack
  - MySpace.com virus
- Web-based attacks can yield the same results as a traditional attack does
  - Usernames/passwords
  - Credit card numbers/SSNs
  - Confidential or classified information
- Automated attacks, let alone self-replicating automated attacks, only makes these threats worse
Why These Attacks Happen

• Web applications are complex!
  – Multiple technologies crossing multiple disciplines
• “Oh, that’s not my responsibility.”
  – Website designers
    • Internal and external
  – Programmers
  – Database admins
  – IT infrastructure admins
• The web application security gap
• Design of an application vs. the implementation of that application
Clearing Up Some Myths

- Layer 7 is dominated by very simple protocols
  - FTP, Telnet, SNMP
  - We are only concerned about HTTP, HTTPS and extensions (WebDav)
- Don’t confuse simple with limited!
- People tend to have a lot of misconceptions about web application security
  - SSL
  - Impact of common vulnerabilities like XSS
SSL only = Silly, Stupid Logic!

• “We use SSL, we don’t have to worry about web security”
  – SSL creates an encrypted tunnel between 2 parties. It provides confidentiality, integrity, and authentication
  – SSL takes place at (depending on who you ask) layers 5 or 6 of the OSI model. SSL is not an Application Layer (ie layer 7) protocol
  – All the attacks I will talk about today are Application Layer attacks running over HTTP.
  – Every attack I discuss today will work against an SSL enabled website

SSL does not protect you from most if not all web application attacks!
SSL Can Hurts You!

- “SSL creates an encrypted tunnel!”
- Unless your SSL end point is your IDS/IPS…
- SSL hides attacks against your web server from your IDS/IPS
A Word About XSS Vulnerabilities

- People have a perception that XSS is silly and not dangerous
  - Maybe true 5 years ago
  - Much worse now
  - AJAX, remoting, RegExs, speed and features of browsers
- People have the perception that XSS is difficult to create
  - Very site specific
  - Tedious to craft
  - Lots of trial and error (manipulate parameter, send, repeat)
- XSS creation is very easy to automate. Even when it’s a complex POST or HTTP header attack
- “Metasploit for web apps!”
  - Payload is separated from positioning code to run payload
A Word About XSS Vulnerabilities

- Phuture Of Phishing - Toorcon 7, Sept 2005
Overview of Web Application Worms and Viruses
Web Worms and Web Viruses

• Traditional attacks are still plentiful
• 2005 saw the release of self-replicating programs that automatically find and exploit web application vulnerabilities
• Web Worms
  – Propagates from host to host infecting each one
  – Conventional worms and XSS worms
  – Language independent
  – Somewhat OS independent (depends on vulnerability they exploit)
  – Runs on web servers (as httpd user)
  – Spreads by sending request to vulnerable target that then runs worm
  – Payloads can be pretty much anything
Web Viruses

- Infects different pages or database entries on the same host (like classic EXE or COM viruses)
- Written in JavaScript (possibly Java, Flash, but not viable because of sandboxing technologies)
- Completely OS independent
- Runs inside browser on client
- Simply viewing an infected page with a browser infects new pages
- Payloads are bad, even with DOM restrictions
  - Basic: Cookie-theft, keylogging, screen/form scrapping
  - Advanced: remote control, arbitrary commands as user
Propagation Methods of Worms and Viruses

- Exploits some vulnerability in a web application
- Sends specially crafted request which...
  - Executes code on target, or
  - Injects code into database, or
  - Can be more exotic (simply reflects script to user, cache poisoning)
- All attacks travel over HTTP

Surely that must be easy to detect and stop, right?
Detecting Layer 7 Attacks?

- Besides port 53, port 80 is the most common open port.
- Just turn off 80 at the firewall? Kind of defeats the purpose of running a web application!
- Down to detecting “malicious” activity
  - Most people say “malicious” = !(“normal”)
  - “Normal” is a moving target
    - Types of users change (housewives during the day, teenagers at night)
    - Load changes with time and season (holiday shopping, morning in South Korea, etc)
    - Massive unanticipated traffic escalations (Slashdottings)
Detecting Layer 7 Attacks?

- Normal site use can look like an attack
  - Large POSTs (ASP .NET ViewState), File Uploads
  - People **want** their site to be crawled by automatic programs
    - **Deliberately** design their sites to be robot friendly
    - Massive hits from a small range of IPs is expected
  - Large sites expect hits from all over the globe
    - IPs from anywhere are expected
    - Complex forms/parameters with funny names or international characters
  - AJAX plays havoc with HTTP traffic filters (Base64 data, etc)
  - “End-to-end” Internet is gone: proxies/NAT are common
  - Anonymity enhancements, other User-Agents break state
Detecting Layer 7 Attacks?

- IDS/IPS evasion is easier at Layer 7
  - Packet-based vs. stream-based analysis
    - Robert Graham’s excellent Toorcon 7 presentation
  - Encoding craziness (URL encoding, UTF-8, etc)
    - A period (".") can be encoded as %2E, %C0%AE, %E0%80% AE, %F0%80%80%AE, %F8%80%80%80%AE, %FX%80%80%80%80%AE.
  - IDS/httpd IP fragment hanging
    - Due to differences in how long IDS holds IP fragments vs. destination TCP/IP stack, IDS and destination see completely different HTTP requests!
    - Dan Kaminski is The Man!
How Does Web Malware Send Attacks?

• Conventional web worm
  – Executing code on the server, anyway you want!
  – Perl::LWP, Sockets, even netcat, curl, wget!
• XSS web worm, web virus
  – Restricted by JavaScript, but not by much
  – Unidirectional (from host to target) a.k.a. “blind requests”
    • Arbitrary GETs to any domain
      – Image objects
      – Script objects
    • Arbitrary POSTs to any domain
      – JavaScript’s createElement builds hidden FORM
      – document.form[0].submit sends the request
How Does Web Malware Send Attacks?

- XSS web worm, web virus (continued)
  - Bidirectional (host and target can talk back and forth)
  - Not just GETs and POSTs, but TRACE, HEAD, Webdav?
    - Arbitrary HTTP to the **same** domain
      - AJAX
    - Arbitrary HTTP to **any domain on the Internet**!
      - SPI Labs has found many covert channels in JavaScript
      - These can be exploited for bidirectional communications with any host on the Internet
      - We will be publishing more soon
Web Application Worms
Web Application Worms (Detailed)

• Two types, conventional (seen in wild) and XSS (theoretical)
• Conventional web worm
  – Real, in the wild threat (Perl.Santy, variants)
  – Run on/by underline OS of the server
  – Almost in all languages: Perl, Python, interpreted languages allows for some OS independence (payload tends to be OS specific)
  – Exploits vulnerabilities in target host’s web applications that allow remote code execution
    • SQL injection (gets database to execute code)
    • Poorly written PHP/Perl/Python/CGI scripts
    • Buffer overflows
Web Application Worms (Detailed)

• Conventional web worm (continued)
  – Finding new hosts to infect
    • Search web application code for references (10.*.*.* IPs!)
    • Ask a 3\textsuperscript{rd} party (search engines, botnet, IM robot, etc)
  – Payload and propagation
    • Already can execute arbitrary code on server for payload
    • Sends requests with attack string to new hosts
  – Limitations
    • User account of exploited web application or web server
    • Underlying OS (chroot isolation, allowed scripting, etc)
Web Application Worms (Detailed)

- XSS web worm
  - Theoretical (MySpace.com attack was a web virus)
  - Runs inside the browser on the client (JavaScript, VBScript)
  - Exploits XSS vulnerabilities to run malicious script
  - XSS vulnerabilities are laughably common!
  - Payload and propagation:
    - Payloads are nasty and advanced (see previous)
    - Sends blind requests to infect backend databases of other hosts (forums, profiles, news stories, etc)
    - Victims view infected page in browser, script executes…
  - Limitations
    - Few imposed by JavaScript, DOM, but they don’t matter
Web Application Viruses
Web Application Viruses (Detailed)

- Real, in the wild threat (MySpace.com virus, Yamanner “worm”)
- Backend databases for dynamic content are injected with XSS
- XSS code served with page, browser executes XSS which launches payload, infects more pages on same host
- Is “virus” the correct term?
  - Infects pages/databases on same host
  - Each infection increasing exposure of virus, runs more often
  - Cannot spread without host “program” (HTML, dynamic content, etc)
- Payloads
  - Geared more towards information stealing and destruction
  - Limitations actually prevent most host damage
Implications of a Web Virus

• Huge! Virus runs in any modern web browser
• Truly cross platform instead of carrying multiple payloads for multiple platforms
• Immune to conventional virus detection
  – Virus stored in database with other highly dynamic content
  – Anti-virus tools work on files, not text snippets
  – Anti-virus tools have file system hooks, not database hooks
  – Server file system, code paths, and binaries are never modified
Implications of a Web Virus

- Immune to any kind of “bad JavaScript” filter
  - Filters would have to be client-side; how does your client-side browser determine what is malicious JavaScript code?
  - To client browser, pages and script come from same legitimate origin (the web server)
  - Same problem as detecting “malicious” HTTP traffic
  - Malicious JavaScript looks just like regular JavaScript
    - Requests images, possibly from multiple, external domains (images.domain.com, blah.adserver.com)
    - Requests scripts from other domains (“link” ads)
    - Manipulates and modifies the DOM tree
    - Hooks OnEvents
Implications of a Web Virus

- Think I’m just selling fear? Compare traditional information stealing Trojan with a web application virus
- Consider a web virus that uses JavaScript to capture keystrokes and send them to a 3rd party
- Has infected a shared calendar page on a web-based CRM
- Any user viewing an infected page gets their calendar page infected (AJAX, blind POST, etc), spreading the virus
- One page view causes spreading; keylogger payload executes and can persist across all of CRM app, even uninfected pages like web-based email (see XSS-proxy, iframe remoting, etc)
- Integrity checks all pass because binaries are unmodified, hooks are intact, no cloaked processes or IPC, and user’s browser is not modified. Works on all platforms, even PDAs!
- No trace of the virus other than occasional info leak to outside
Analysis of Perl.Santy
Analysis of Perl.Santy

• Conventional web worm (many variants)
• December 2004 – Spring 2005
• Perl with LWP, Sockets (varies)
• Attack vector: Exploits phpBB highlighting bug for code execution by specially crafted input parameters
• Propagation:
  – Google searches with static string to find vulnerable hosts
  – GET requests with attack string, propagating virus
• Payload
  – Trivial page defacement of all html, php, etc documents
Analysis of Perl.Santy

- Google search string provided choke point
- Static search strings stored inside the Perl source code
- Host selection algorithm extremely poor
  - Pick a ccTLD
  - Pick a version of phpBB.

Google Error

We're sorry...

... but we can't process your request right now. A computer virus or spyware application is sending us automated requests, and it appears that your computer or network has been infected.
Analysis of Perl.Santy

- No mutation of source code, search string, or attack string
- Payload was silly
Analysis of MySpace.com Virus
Analysis of MySpace.com Virus

• Web virus
• October 2005: Infected 5th largest domain on the Internet
• JavaScript with AJAX
• Attack vector: XSS exploit allowed <SCRIPT> into user’s profile
• Propagation:
  – Used AJAX to inject virus into the user profile of anyone who viewed an infected page
• Payload:
  – Used AJAX to force viewing user to add user “Samy” to their friends list
  – Used AJAX to append “Samy is my hero” to victim’s profile
Filtering Input Is Hard!

- MySpace.com did a very good job filtering certain words `<SCRIPT>`, JavaScript, `innerHTML`, certain characters like “
- No `<SCRIPT>` not good enough
  - `<DIV style="background('javascript:whatever')">`
- Whitespace is your friend
  - `tag.inne` + `rHTML` `java\nscrip` `t` `String.fromCharCode()`
- God bless the `eval` statement
  - Parses and executes JavaScript stored in a string
  - `String` doesn’t have to be defined in JavaScript. Can be in the DOM
  - `<DIV id="code" expr="alert('xss')"
    style="background('java\nscrip:eval(document.code.expr)')">`

See [http://namb.la/popular/tech.html](http://namb.la/popular/tech.html) for all technical challenges
Infection Method Explained

profile.myspace.com  www.myspace.com

Switch Domains

GET Confirm page (hash)

Extract hash, POST payload
Analysis of MySpace.com Virus

- Awesome hack! No, I didn’t write it.
  - I did present about XSS + AJAX attacks at Toorcon 7, a month before the virus hit
- Excellent proof of concept about how using AJAX is a security risk even though it obeys the DOM security model
  - Web server cannot tell the difference between AJAX requests and web browser requests
- Shows how AJAX + JavaScript RegExs can handle complex login sequences spanning multiple pages
- MySpace lucked out as it could have been much worse
Yamanner Web Virus
Analysis of Yamanner Virus

- Web Virus + traditional worm = ???
- June 2006: Infected and took down 3\textsuperscript{th} largest web email provider Yahoo
- JavaScript with AJAX
- Attack vector: Allowed JavaScript into email message
- Propagation:
  - Used AJAX to send infected email to all Yahoo users in address book
- Payload:
  - Stole entire address book
  - (Attempted) to pop up an ad window

\textit{Let's look at the Source code shall we?}
Analysis of Yamanner Virus

- MySpace.com virus to Yamanner Virus = 8 months
- Proof of concept to criminal enterprise
- Yahoo should be ashamed… onload… come on!
Hypothetical, Worse Case Examples of Web Malware
Now put on your Black Hats!

• The Perl.Santy worm and MySpace.com virus were real world examples of concepts that web security people have talked about for years
• Both had very childish payloads
• So, what is a worst case scenario with these types of threats?
• Next, I present you with two hypothetical and truly evil examples of extreme web malware
  – Swogmoh Web Worm
  – 1929 Web Virus
Swogmoh Web Worm (Details)

- "HOly MOther of GOD, We’re SCrewed!” backwards
- Written in Perl::LWP
- Attack vector: Multiple SQL injection vulnerabilities of different web applications
- Propagation:
  - Use Google to locate new sites vulnerable to one of our SQL injection vulns
  - Mutate our search string to avoid bottlenecks
    - Allinurl: ~= inurl:
    - Add ignored words (the, in, of, at, a, an) or repeat words
    - Algorithm to generate English words or /usr/local/dict
    - Word order
Swogmoh Web Worm (Details)

- Propagation (continued)
  - If I don’t get a results page, Google can detect search string
  - Randomly select next search engine
- Mutate virus source code
  - Interpreted scripts are easy to mutate
    - New comments, etc
    - Replace control structures (do: while = while, while = for, if-then-else = switch)
    - Encrypt the static strings with a different dynamically generated key **per copy**!
    - Perl is text parsing king. Complex text replacement is no big deal.
Swogmoh Web Worm (Details)

• Payload
  – Keeps track of successful infections by trying to GET magic page from victims. After 100 successfully infections, launch payload!
  – Known vulns = known apps = known database structures
    • Dump usernames/passwords to mailing lists, blog comments, or Slashdot so you can retrieve them
    • Or listen to the sound of 100,000 DROP TABLEs
    • Or INSERT INTO databases with garbage
  – Flood email systems, webs servers of major anti-virus companies, app creators to slow their response
Swogmoh Web Worm (Details)

• Impact and improvements
  – Will vary but generally very bad
  – Defeated by backups
  – Google might be able to filter search strings faster than anticipated, but that’s why we have multiple search engines
  – Balance between number of hosts infected and payload must be research to ensure maximum possible infections
  – Works with any remote code execution vulnerability
    • Abstract virus code from remote code execution code
    • Pluggable interface for new vulnerabilities
    • Start virus with multiple vulnerabilities
1929 Web Virus (Details)

- Infects major stock trading site (E-trade, Ameritrade, etc)
- JavaScript with AJAX
- Attack vector: XSS exploit to get `<SCRIPT>` into forum, user profile, stock ticker, etc
- Propagation:
  - Uses AJAX/blind POST to inject script into other pages using credentials of any user viewing infected page
- Payload:
  - Uses AJAX to place buy and sell stock orders on your behalf
  - Complex confirmation pages are not an issue (short of a Captcha)
1929 Web Virus (Details)

- Two modes
  - Online mode: virus decisions controlled by 3rd party
  - Research mode: virus makes buy/sell decisions by itself
- Online mode
  - Use iframe heartbeating (see XSS-Proxy) to send commands from external 3rd party to infected pages running in browsers
  - Inflict damage to stock market as thousands of users sell otherwise healthy stocks
  - Damage 1,000 individual portfolios simultaneously by buying junk stocks
• Research mode
  – Selects stocks to monitor
    • Randomly build stock symbols
    • Price/Earnings ratios
    • Trade volume thresholds
  – AJAX used to sample these stocks at set intervals
  – Calculate rate of change of stock price to find buying/selling trends
  – When rate of change approaches zero we are nearing the top or bottom of a trend curve, sign tells us which direction
  – Buys stocks at the highest prices
  – Sells stocks at lowest prices
1929 Web Virus (Details)

- **Impact**
  - Try explaining to the SEC that you really didn’t make a trade
    - It came from your IP
    - You were online
    - Trades mixed in with other legitimate trades
  - Eventually stock trading site will find virus, remove it, and attempt to sort real trades from virus trades
  - Does not really matter in the end
  - External brokers will have made trade decisions based on effects of the virus’ trades. The virus has affected the entire stock market.
Guidelines for Writing Secure Web Applications
Guidelines for Writing Secure Web Applications

- Ultimately, web malware occurs because of vulnerabilities in web applications
- Fixing the vulnerabilities stops both aspects of web malware
  - Initial injection and further propagation
  - Payload execution
- Your web applications are the bricks in the walls of your website. Do you really trust a brick you downloaded from SourceForge?
- 90% of web application security is validating user input
Guidelines for Writing Secure Web Applications

• *Never trust anything you get from the client!*  
• Everything can be modified  
  – “Hidden” HTML input tags  
  – Cookies  
  – URL parameters  
  – POST data  
  – HTTP headers
Guidelines for Writing Secure Web Applications

• Never use input you get from the client without sanitizing it
  – Enforcing data types
    • Only numbers?
    • Only letters?
    • Formatting (credit cards, telephone numbers)
  – Length restrictions (TinyDisk file system)
  – Escaping characters like <“‘|;> to avoid SQL injection and XSS attacks
  – PHP/ASP all have built in functions for this. A well placed RegEx can stop most attacks.
Guidelines for Writing Secure Web Applications

• Input validators should be implemented on both sides of a web application
  – Client-side validation should exist **solely** for performance issues
  – Server-side validators are the only way to enforce any limits
• Frontend code should properly represent backend code
  – Backend code for an HTML FORM that uses POST should only read values that were posted. (Request.Form vs. Request.QueryString, etc)
• Over engineering is very bad
• Applications should only provide enough functionality to work
  – If you have static content, do not use scripting technologies (ASP, PHP, JSP, etc) to serve it
  – LDAP directory vs. full SQL-driven relational database
Summary

• Web application malware is no longer theoretical.
• So far web malware payloads have been silly. Expect this to change.
• Web malware payloads can equal traditional malware in terms of damage and information leakage.
• Web malware operates on a different level than traditional malware. Defenses are not as readily available for these threats.
• For these reasons, web malware is actually more dangerous than traditional malware.
• Popularity and buzzwords are driving uneducated programmers into web application development, making the problem worse.
• Properly securing web applications inputs will stop most web malware.
Questions?
Analysis of Web Application Worms and Viruses

Billy Hoffman (bhoffman@spidynamics.com)

SPI Labs Security Researcher