



Covert Crawling: A Wolf Among Lambs

Billy Hoffman (bhoffman@spidynamics.com)

SPI Labs Security Researcher

Overview

- State of web application attacks
- How does a covert crawler change things?
- Obstacles to create a covert crawler
 - Acting like a browser
 - Acting like a human
 - Throttling and timing issues
- Implementation of a covert crawler
- Questions

The State of the Union

- People are hacking websites because it's easy
 - Web vulnerabilities like SQL Injection, session hijacking, cookie theft and cross site scripting are laughably common
- People are hacking websites because it's news
 - MySpace.com XSS virus (October 2005)
- People are hacking websites because they can make money
 - Identity theft, credit card numbers, content theft

The State of the Union

- Administrators are patching some holes (php[whatever] exploit of the week)
- Relying on logs if they get attacked
- Contact IPs of the attacker, normally proxies that shouldn't be open, and then two people go after attacker
- Whether you can successfully sneak an attack by an IDS/IPS is debatable, but no one just attacks out of thin air
- Almost all attacks are foreshadowed by some form of reconnaissance
- Administrators rely on server logs to find this initial probing as well

Why Performance Reconnaissance

- Running a straight audit of an entire web application is not practical if you are malicious
 - Takes too long
 - Too much network noise (IDS evasion or not)
- Reconnaissance provides information to assist an attack
 - What versions of what technologies are used
 - Structure and layout of site
 - State keeping methods, authentication methods, etc
 - HTML comments
 - Developer names
 - Contracted company names
 - Email addresses
 - Provides a subset of pages to actually attack

Types of Reconnaissance

- Browse site by hand in a web browser
 - Lets the user direct the search at what they want
 - Looks for areas with specific vulnerabilities (SQL Injection in search engine, ordering system, etc)
 - Takes time user could be doing other things with
 - Not exhaustive search of entire site
- Automated crawler
 - Wget, some custom Perl::LWP script
 - Hits every page on the site
 - Very obvious in server log who crawled them, when, and what they got

Covert Crawling

- Covert crawling has all the pros of regular automated crawling (exhaustive search, automatic, saves resources for later analysis)
- Covert crawling uses various tricks to make the crawl appear as if it actually was hundreds of different users from different IPs that were browsing the target website
- Not used to actually attack a site.
 - Finds a likely subset of pages to attack
 - Attacker can later use IDS/IPS evasion techniques to launch attacks on a subset of pages
- Logs are unable to show a reconnaissance ever happened!

Why Run a Covert Crawl?

- Malicious user performing reconnaissance who wants to reduce the forensics trail
- A company who wants to monitor a competitor's progress without leaving a trace. Prevents their research interests from being leaked based on their crawling/browsing (IBM patent database)
- A lawyer monitoring a website on behalf of a client for libel statements or posting
- Miscellaneous intelligence gathering
- General privacy/anonymity reasons

Using a Search Engine Cache?

- *“This is silly, use the Google cache!”* Very bad idea.
- HTML comes from Google cache, but IMGs, OBJECTS, SCRIPTs come for target site. Server logs will reveal Google cache referrers.
- Google respects the robots exclusion standard
- Google doesn't cache everything (external JavaScript)
- Things can be removed from the Google cache
- Limited to 1000 queries a day if using Google API
- Google does not serve > 1000 results per query
- Limited by Google allowing you use the cache in the future
- Crawling the site yourself is the only way to guarantee you download all content both now and in the future

Obstacles to Overcome

- Covert crawler consists of requester controlled by a master program
- Needs to mimic a browser controlled by a human
- Not an easy task
 - Crawlers don't act like browsers
 - Crawlers and humans make fundamentally different choices regarding which links to follow when
- Multiple IPs must be used to spread the crawl out to reduce the bandwidth footprint from any single IP address
 - Must control all these threads
 - Must maintain proper session state *for each thread*
 - Must prevent threads from appearing to be collaborative in any way
- The crawler must be throttled to not overwhelm a site

Obstacle 1: Acting Like a Browser

Behavior: Crawlers vs. Browsers

- Crawlers never visually render responses, so HTTP headers describing content abilities are minimal
- Browsers are rich *user-agents* which sends many HTTP headers to get the best possible resource
- Crawlers request HTML and parse it to find more links
- Browsers request HTML and all supporting files
- Crawlers are relatively simple
- Browsers are complex. They contain code for running Java applets, Flash, JavaScript, and ActiveX objects. These technologies can make direct HTTP connections themselves

Details of a Browser Request

- Browsers send lots of HTTP headers with each request
- A covert crawler must duplicate the order and values of these headers
- Pick the most common browser, the most common version

▼ Hypertext Transfer Protocol

▶ GET / HTTP/1.1\r\n

Accept: */*\r\n

Accept-Language: en-us\r\n

Accept-Encoding: gzip, deflate\r\n

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Wind

Host: www.google.com\r\n

Connection: Keep-Alive\r\n

Cookie: PREF=ID=b99026fb5a4bd73c:TM=1135629271:LM=1

\r\n

Manipulating HTTP Headers in Java

- Java's HTTP functions are not useful
 - Proxy nastiness
- HttpClient – full featured HTTP library for Java
- Ronald Tschalär
- <http://www.innovation.ch/java/>
- Allows direct access to HTTP headers
 - Can control ordering (with some hacking)
 - Can control types and values

```
try
{
    //blindly accept cookies
    CookieModule.setCookiePolicyHandler(null);

    if(http == null)
        http = new HttpURLConnection(url.getHostname());

    PBrowser.configHTTPHeaders(http);

    HTTPResponse response = http.Get(url.getResource())
    int code = response.getStatusCode();
    LogManager.AppendLog(log, code + " " + url.toString());

    byte [] data = null;

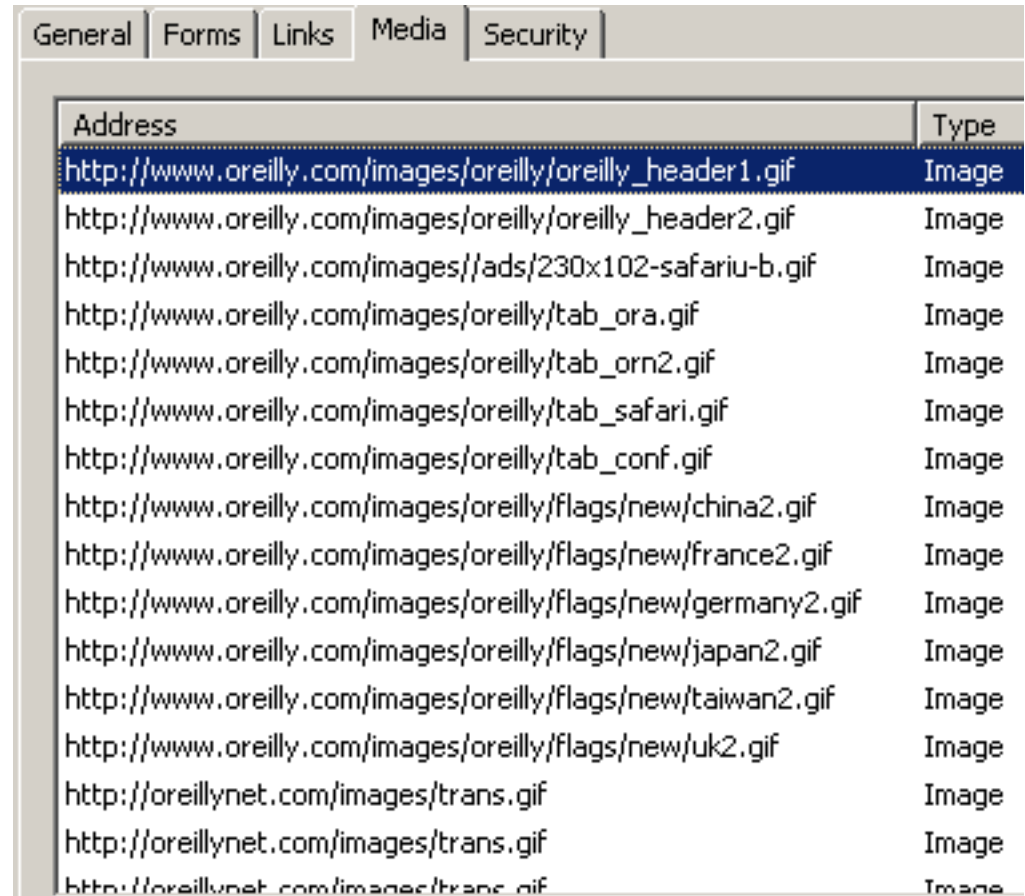
    String t = response.getHeader("Content-Type");
    if (t != null)
        LogManager.AppendLog(log, "Content-Type is " +

    if(code >= 200 && code < 300) {
        data = response.getData();
        DataStorage.addFile(url, data);
    }

    //is it a HTML file?
    if(t.toLowerCase().equals("text/html") && data !=
        //create an HTMLPage from the data
```

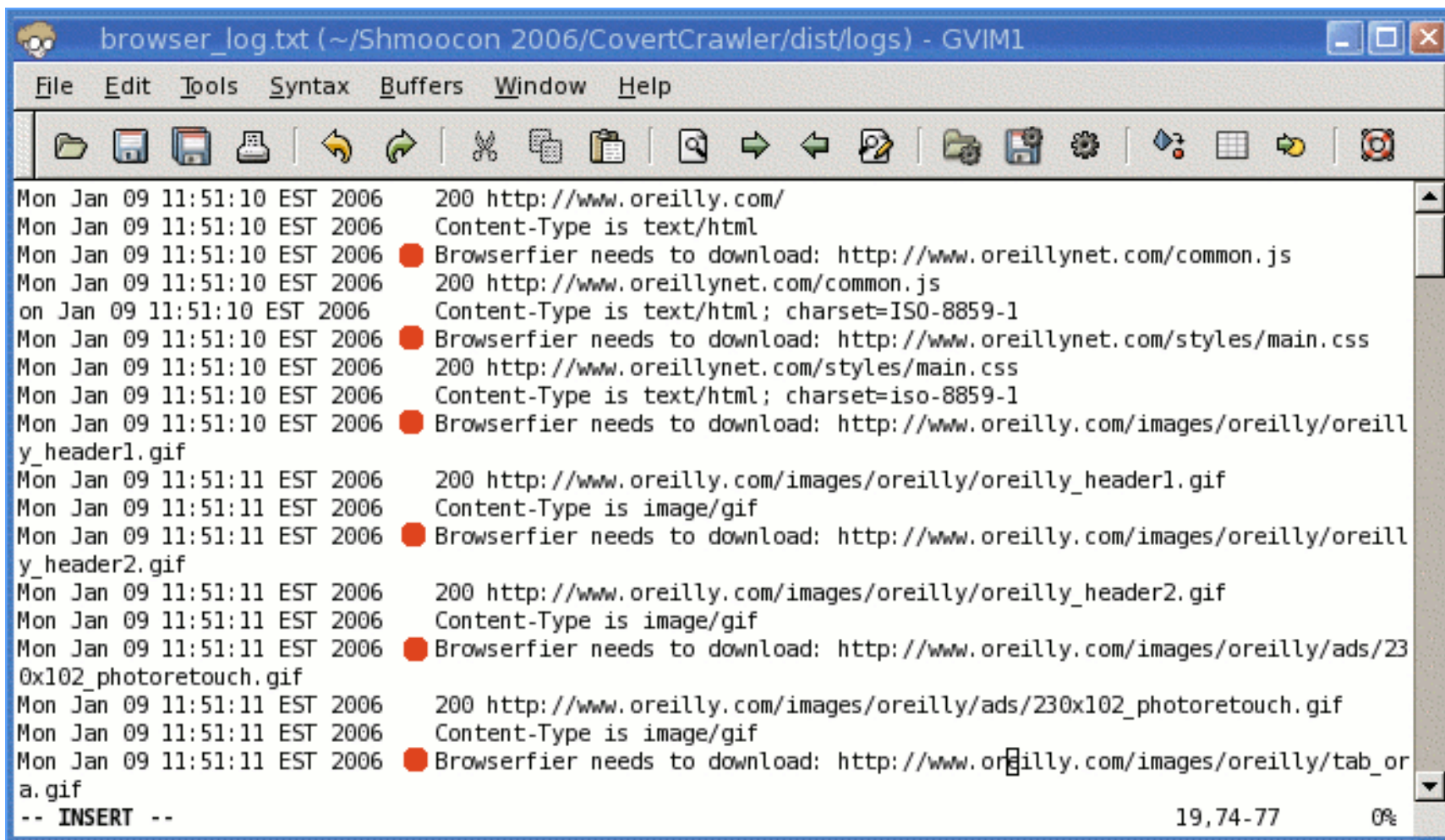
Getting Page Resources

- One HTML page causes several other HTTP transactions
- Images
- Image link maps<MAP> and <AREA>
- Client-side scripting <SCRIPT>
- Complex media <OBJECT>, <APPLET>, and <EMBED>
- Favicons <LINK>



Address	Type
http://www.oreilly.com/images/oreilly/oreilly_header1.gif	Image
http://www.oreilly.com/images/oreilly/oreilly_header2.gif	Image
http://www.oreilly.com/images//ads/230x102-safariu-b.gif	Image
http://www.oreilly.com/images/oreilly/tab_ora.gif	Image
http://www.oreilly.com/images/oreilly/tab_orn2.gif	Image
http://www.oreilly.com/images/oreilly/tab_safari.gif	Image
http://www.oreilly.com/images/oreilly/tab_conf.gif	Image
http://www.oreilly.com/images/oreilly/flags/new/china2.gif	Image
http://www.oreilly.com/images/oreilly/flags/new/france2.gif	Image
http://www.oreilly.com/images/oreilly/flags/new/germany2.gif	Image
http://www.oreilly.com/images/oreilly/flags/new/japan2.gif	Image
http://www.oreilly.com/images/oreilly/flags/new/taiwan2.gif	Image
http://www.oreilly.com/images/oreilly/flags/new/uk2.gif	Image
http://oreillynet.com/images/trans.gif	Image
http://oreillynet.com/images/trans.gif	Image
http://oreillynet.com/images/trans.gif	Image

“Browser-fying” a Crawler



```
browser_log.txt (~/Shmoocon 2006/CovertCrawler/dist/logs) - GVIM1
File Edit Tools Syntax Buffers Window Help
Mon Jan 09 11:51:10 EST 2006 200 http://www.oreilly.com/
Mon Jan 09 11:51:10 EST 2006 Content-Type is text/html
Mon Jan 09 11:51:10 EST 2006 ● Browserfier needs to download: http://www.oreillynet.com/common.js
Mon Jan 09 11:51:10 EST 2006 200 http://www.oreillynet.com/common.js
on Jan 09 11:51:10 EST 2006 Content-Type is text/html; charset=ISO-8859-1
Mon Jan 09 11:51:10 EST 2006 ● Browserfier needs to download: http://www.oreillynet.com/styles/main.css
Mon Jan 09 11:51:10 EST 2006 200 http://www.oreillynet.com/styles/main.css
Mon Jan 09 11:51:10 EST 2006 Content-Type is text/html; charset=iso-8859-1
Mon Jan 09 11:51:10 EST 2006 ● Browserfier needs to download: http://www.oreilly.com/images/oreilly/oreilly_header1.gif
Mon Jan 09 11:51:11 EST 2006 200 http://www.oreilly.com/images/oreilly/oreilly_header1.gif
Mon Jan 09 11:51:11 EST 2006 Content-Type is image/gif
Mon Jan 09 11:51:11 EST 2006 ● Browserfier needs to download: http://www.oreilly.com/images/oreilly/oreilly_header2.gif
Mon Jan 09 11:51:11 EST 2006 200 http://www.oreilly.com/images/oreilly/oreilly_header2.gif
Mon Jan 09 11:51:11 EST 2006 Content-Type is image/gif
Mon Jan 09 11:51:11 EST 2006 ● Browserfier needs to download: http://www.oreilly.com/images/oreilly/ads/230x102_photoretouch.gif
Mon Jan 09 11:51:11 EST 2006 200 http://www.oreilly.com/images/oreilly/ads/230x102_photoretouch.gif
Mon Jan 09 11:51:11 EST 2006 Content-Type is image/gif
Mon Jan 09 11:51:11 EST 2006 ● Browserfier needs to download: http://www.oreilly.com/images/oreilly/tab_or
a.gif
-- INSERT -- 19,74-77 0%
```


More Browser Emulation

- Acting like a browser more than just sending the same HTTP requests
- How you send them
 - Use HTTP persistent connections
 - Use pipelining if applicable
- If you even need to send them
 - Implement a crawler cache
 - Respect cache control headers
 - Should only request a resource if not already cached
 - If a resource has a cache directive use conditional GETs to attempt to retrieve it if it is seen again.
 - Conditional GETs keep the server logs looking normal

Complex HTML issues

- Some features of HTML make “browser-fying” difficult
- HTML can contain objects that make HTTP connections on their own
 - JavaScript has AJAX (HTTP connection back to origin)
 - Flash can use sockets
 - Java applets can use sockets
 - ActiveX can use sockets
- META refresh tags
 - Other objects continue download, then redirect happens

Writing Java, Flash, JavaScript, and ActiveX parsers is outside the scope of this project. I ignore them for now.

Obstacle 2: Browsing Like a Human

Behavior: Crawlers vs. Humans

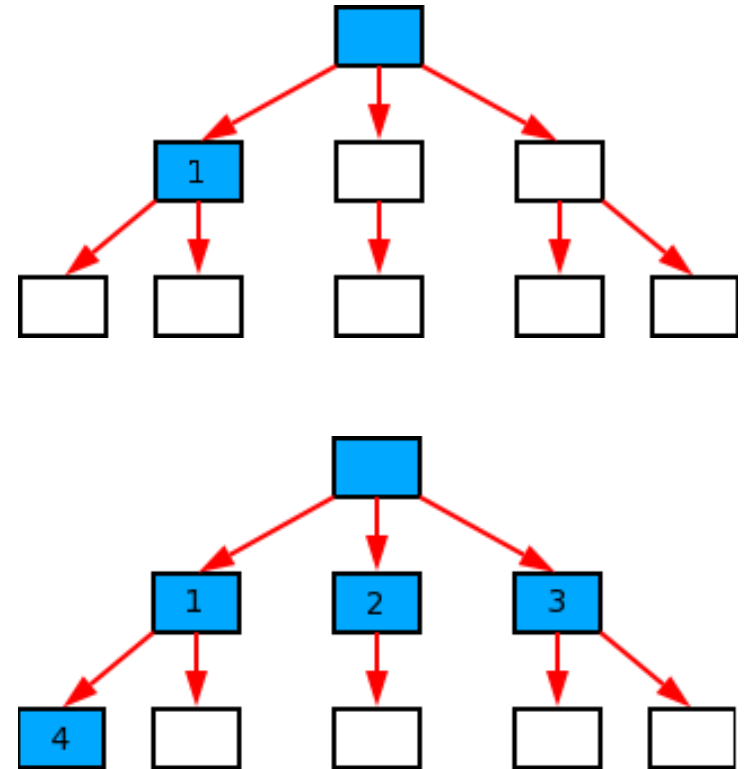
- Crawlers navigate links and make requests in a very obvious and predictable pattern. This is easy to detect in logs
- Humans make browsing decisions based on content and other factors. The request pattern is very different from a crawler's
- Title really should be “Behavior: Computers vs. Humans”
- Replicating human behavior is a complex problem.
- This project was not a master thesis on AI
- We need to find a *good enough solution* that is practical.

Pseudo code of a Breadth First Crawler

- Traditional breadth first search (BFS) crawler
 1. Remove a link from the pending queue
 2. Retrieve the resource
 3. Store local copy for later analysis (indexing, etc)
 4. If Content-Type is not *text/html* go to step 6
 5. For each link in the HTML
 - Check if link already exists in pending link queue
 - If not, add link to end of pending link queue
 6. If pending link queue is empty, stop the crawler
 7. Go to step 1

Page Retrieval for a BFS Crawler

- Queue data structure causes crawler to visit all the pages of *equal depth* before going deeper
- Pros
 - Simple design
 - Lots of examples
- Cons
 - Very obvious, non-human request order
 - Pages will be requested when the previous response contained no link to that page



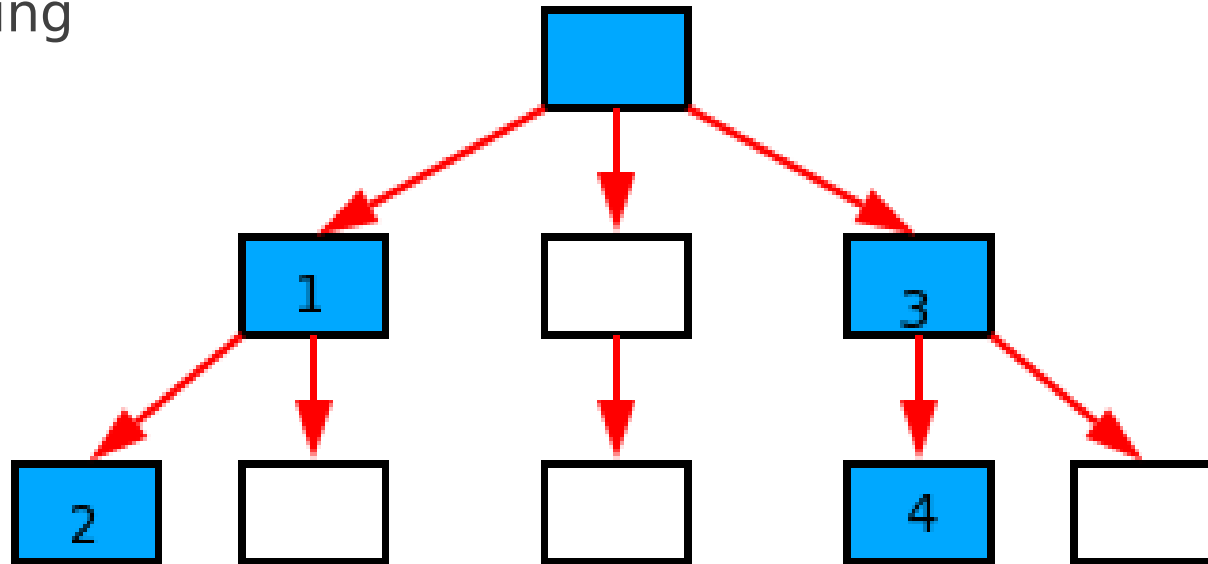
Browsing like a Human

- All links are equal...
- ...but some links are more equal than others.

The screenshot shows the O'Reilly website homepage in a Mozilla Firefox browser. The browser's address bar shows 'http://www.oreilly.com/'. The page features the O'Reilly logo (an owl) and the text 'O'REILLY computer books • conferences • online publishing'. A green box highlights the logo and the text 'O'REILLY'. Below the logo, there is a navigation bar with links like 'Book List', 'Learning Lab', 'Documents', 'O'Reilly Store', 'Newsletters', 'Press Room', and 'Jobs'. A search bar is located on the left side. A green box highlights the search bar and the 'Go' button. The main content area is divided into several sections. The 'Featured New Releases' section includes a book titled 'Adobe Creative Suite 2 Workflow' with a green box around the title and a description: 'With full-color examples and dozens of practical tips and tricks, this book is the working professional's guide to taking advantage of that CS2 has to offer. Its concise explanations and step-by-step exercises teach you the core skills and techniques to create an integrated workflow, which not only saves you time, but will give you more flexibility to move between applications.' The 'Complete C Reference' section includes a book titled 'C in a Nutshell' with a green box around the title and a description: 'This long-awaited book is the complete reference to the C programming language and C runtime library. It covers virtually everything you need to program in C, describing all the elements of the language and illustrating their use with numerous examples. Meant to serve as a convenient, reliable companion in your day-to-day work as a C programmer.' The 'Special Offer' section features a 'Scale Your Business with MySQL' promotion. The 'News & Articles' section includes a link to 'Pioneer Podcasters Share Insider Tips, Techniques & Equipment'. The 'O'Reilly Staff Picks' section features a book titled 'High Performance MySQL' with a green box around the title and a description: 'Anyone looking to understand the internals of MySQL and truly problem solve with the server would be a fool not to read High Performance MySQL, cover to cover at least once. For a moderately small book, it packs an amazing punch. Devoted users are able to get the most out of this book while beginners can use it as a wake up call to how far they can take their learning. This book does a wonderful job of singling out key parts of the internals in order to design better databases and queries. While MySQL continues to evolve, the information in this book will stay relevant for a number of years. Read more: Ryan Grimm, Software Developer, Information Systems'. A green box highlights the title 'EMERGING TELEPHONY You Ain't Seen Nothing Yet' and the description: 'Developers now have the right tools and the right new desktop applications, telephone services, and'. The browser's status bar at the bottom shows 'Done' and 'AdBlock'.

Page Retrieval for a Human

- Pattern of human requests are different than BFS crawlers' requests
- Each humans will access resources in a *different* order based on personal tastes. Crawlers almost always act the same.
- Lesson: link selection is extremely important in covert crawling



Browsing Like a Human

- Humans and crawlers see HTML differently
 - Crawlers simply see a list of links
 - Humans see a page with content and *filter* the links on the page
- Humans filter links using several factors
 - Positive link context: Is it something the user is interested in?
 - Negative link context: Is it something the user is not interested in?
 - Link presentation: Is the presentation of the link itself interesting enough to warrant clicking it?
- How can a crawl filter links like a human does?

Filtering links

- Understanding the context of a link is complex
 - We can fake it by examining the contents of the link's text
- Understanding the presentation of a link is easy
 - Defined by HTML structure, CSS, etc
 - Cannot evaluate images. Could say "*Never ever click on me!*"
- Doesn't Google have to deal with this? Page Rank, link relations, scoring of links?

Link Scoring

- Score for each link is calculated, defines how “popular” a link is
- Pretty straightforward
 - Contains emphasis relative to surrounding text
 - Length of link text, both word size and word count
 - Rate the “goodness” or “badness” of the link text words
 - Good words: new, main, update, sell, free, buy, etc
 - Bad words: privacy, disclaim, copyright, legal, about, jobs, etc
 - For images, calculate the image’s area proportionally to a 1024x768 screen
 - Also score *alt* attribute text if present

Link Scoring: oreilly.com

- Works fairly well!

```
Working on http://www.oreilly.com/
+5 IMG tag detected
+57 IMG area bonus
Working on http://www.oreilly.com/
+5 IMG tag detected
+39 IMG area bonus
+4 word bonus (shop)
+4 word bonus (hot)
Working on http://oreilly.com/
+5 IMG tag detected
+2 IMG area bonus
Working on http://www.oreillynet.com/
```

```
Working on http://www.oreillynet.com/cs
+2 long link text bonus
+4 word bonus (new)
Working on http://press.oreilly.com/
+3 for multiple words
Working on http://jobs.oreilly.com/
-5 word punishment (jobs)
Working on http://www.oreilly.com.cn/
+5 IMG tag detected
```

```
63 http://www.oreilly.com/
36 http://www.oreilly.com/cata
er8tmm
28 http://safari.oreilly.com/
25 http://www.oreilly.com/cata
25 http://www.oreilly.com/cata
23 http://www.oreillynet.com/p
html
22 http://learninglab.oreilly
22 http://www.oreillynet.com/p
22 http://www.oreilly.com/cata
21 http://events.oreilly.com/p
20 http://www.linuxdevcenter.c
20 http://www.oreilly.com/cata
19 http://oreilly.useractive.c
19 http://conferences.oreilly
19 http://www.oreilly.com/cata
19 http://www.oreilly.com/cata
18 http://safari.oreilly.com/
```

Link Scoring: oreilly.com

The screenshot shows the O'Reilly website homepage. At the top, the O'Reilly logo is prominently displayed with the tagline "computer books • conferences • online publishing". Below the logo is a navigation bar with links for "Home", "About Us", "Contact Us", "Site Map", "Privacy Policy", and "Terms of Service". A search bar is located on the left side of the page. The main content area is divided into several sections:

- Featured New Releases:** A section titled "Create an Integrated Workflow" featuring the book "Adobe Creative Suite 2 Workflow" by Eric Fretwell. The description highlights that the book is a professional's guide to taking advantage of all the CS2 features and synergies, explaining them in a step-by-step manner that teaches the user skills and techniques to create an integrated workflow, which not only saves you time, but will give you more flexibility to move between applications.
- Complete C Reference:** A section titled "C is a Nutshell" featuring the book "C is a Nutshell" by Dennis Flanagan. The description states that this is the complete reference to the C programming language and C standard library, covering virtually everything you need to program in C, describing all the elements of the language and illustrating their use with numerous examples. It is described as a portable, reliable companion to your day-to-day work as a C programmer.
- Scale Your Business with MySQL:** A section titled "Scale Your Business with MySQL" featuring the book "MySQL Users Conference". The description mentions that the MySQL Users Conference is the largest gathering of MySQL developers, a series of DBAs. It is the only event where you will be able to join the core MySQL development team and over 1000 developers, service innovators, and technology partners under one roof, join us in Santa Clara, California on April 24-27. Register before March 6 and save \$200.
- Pioneer Podcasters Share Insider Tips, Techniques & Equipment:** A section featuring the book "Podcasting Hacks" by Ryan Green. The description states that this is the first in-depth, step-by-step guide to podcasting with pioneer podcasters Doug Kaye and James Hahnke. Doug is the founder of IT Conversations, the influential site that features podcasters covering important events, programs, and interviews with industry luminaries. James is the founder of Radio 7, the popular podcast radio show covering all things digital music. Hack is the author of "Podcasting Hacks".
- IP Telephony: You Ain't Seen 'Nuthin' Yet:** A section featuring the book "IP Telephony: You Ain't Seen 'Nuthin' Yet" by Ryan Green. The description states that this is the right tools and the right motivation to build a wide range of new desktop applications, telephone services, and more.

On the right side of the page, there is a section titled "O'REILLY STAFF PICKS" featuring a book by Ryan Green, "Software Developer, Information Systems". The description states that anyone looking to understand the inner workings of MySQL and why performance MySQL power is crucial for your business should pick up an amazing parcel. Software developers and testers alike will find this book a while beginner can use it as a quick guide to how to they can be better serving. The book does a wonderful job of bringing all sorts of the information into a single place, so you can find what you need in one place. The information in this book will stay relevant for a number of years.

Lessons Learned From Link Scoring

- Position in HTML is not a good indicator for score
 - Separating structure and presentation
- Style sheets make emphasis detection difficult
 - Can be declared in multiple places
 - Browsers are forgiving of bad CSS like bad HTML
 - Can overload styles inline inside the HTML
- Bad words are pretty static – who reads *privacy.html*?
- Good words can vary from site to site
- Don't score links whose target is the current page
- *Table of contents* style link lists are hard to deal with

Using Link Scoring for Crawlers

- Use a priority queue for pending links instead of regular queue
- Sort by decreasing link score
- When inserting into queue, check if URL already exists
 - If so, sum the 2 link scores
 - Averaging would actually hurt a link's score. Consider a big image link and a small text link
 - Issues of URL equality with multiple threads (later in slides)
- Front of queue is always the most *popular* link the crawler has seen

Acting Like a Human

- Humans need time to process what the browser presents
- Each crawling thread of the crawler must wait some amount of time before requesting a new page
 - Based on actual size of rendered content
 - Random, using average user-browsing statistics
- Give the different threads *personalities*
 - Give a program some keywords it likes based on site content (like Apple, or Firewire)
 - On pages that have those keywords, crawler takes a longer time *looking* at the page before moving on
 - Other pages the thread quickly clicks through

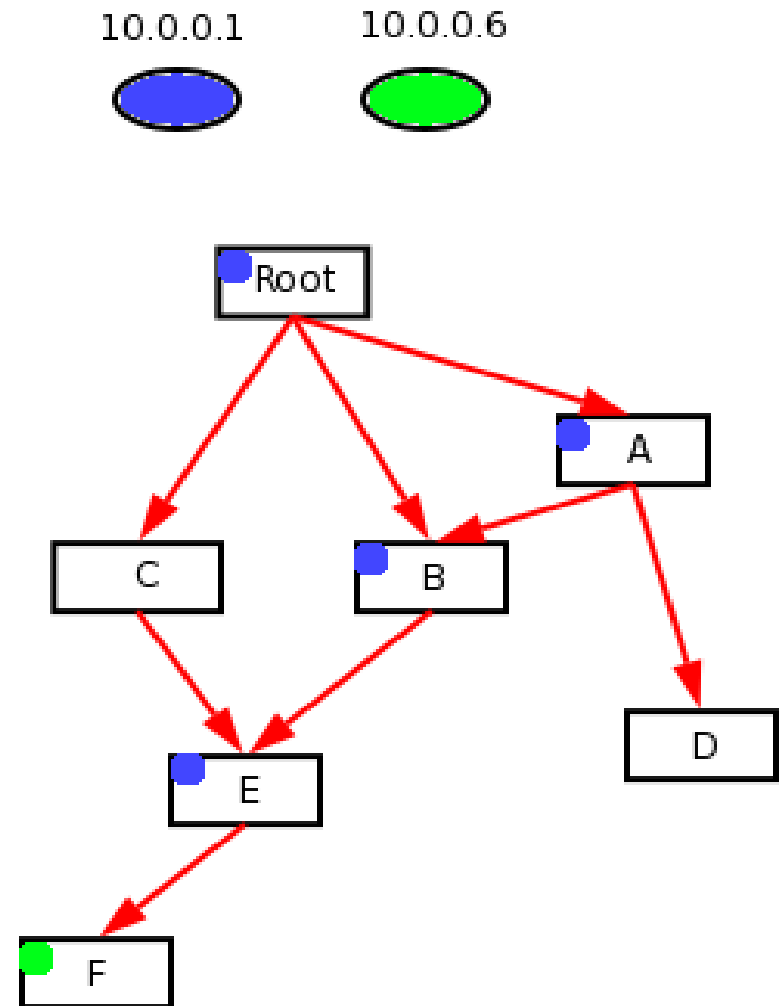
Obstacle 3: Reducing the Bandwidth Footprint

Leveraging Open Proxies

- We need to spread the crawl over multiple IPs. Open HTTP proxies are used
 - Possible a weakness in the covert crawler since we are leaving IP addresses of known proxies in the server logs
- Choice of proxies is very important
 - Crawling a US bank, only use US proxies, some in Western Europe.
- Minimize the number of proxies that announce they are proxies
 - Via, X-Forward-For, Max-Forwards show an HTTP request is being proxied
- Test proxies by examining headers they allow through
- Sometimes it's not so bad to admit you are a proxy

The Deep Link Problem

- Users get to deep pages inside a website 3 ways:
 - Navigated there from some other internal page
 - Followed a bookmark (no Referrer header)
 - Followed a link, usually a search engine (has Referrer)
- When we sending requests across multiple IPs, sending deep requests is dangerous
- Green requests a page they've never seen

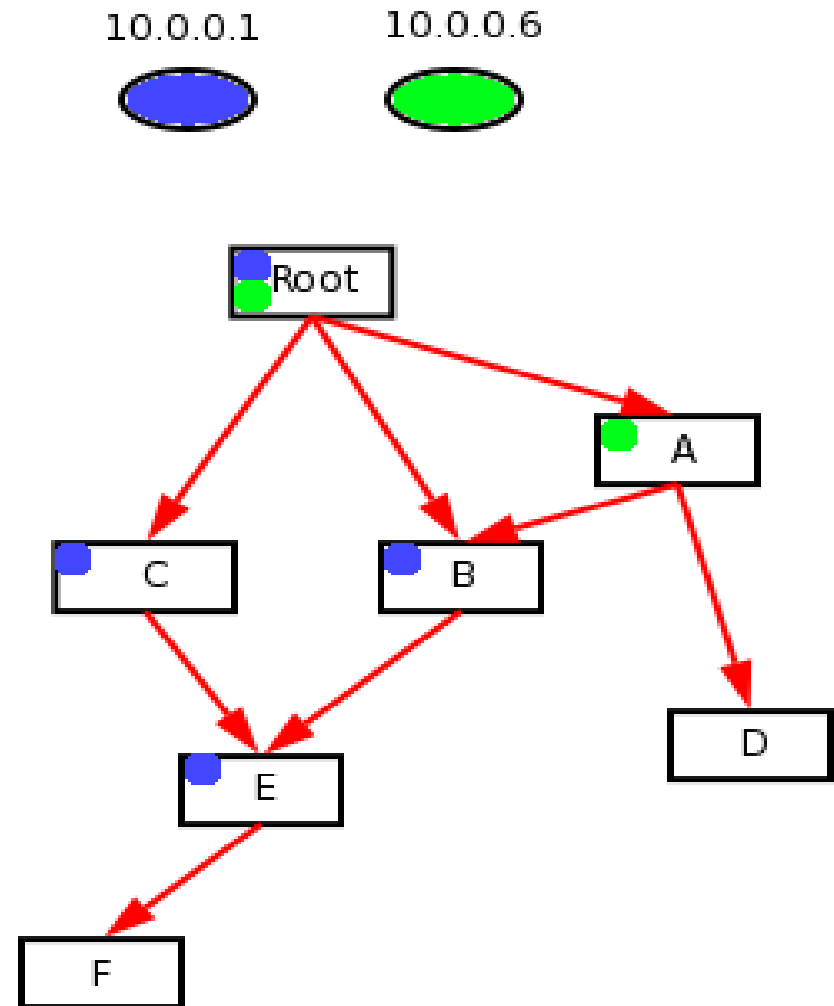


The Deep Link Problem

- IPs that make requests out of the blue look suspicious.
- A covert crawler can never make deep requests from an IP that has no business making that request
- So our instead of telling our crawling threads *“Make a request for [URL]”* we tell them *“Get to this [URL]”*
- We keep a shared graph representing the website
 - Unidirectional edges, cyclic graph
 - Nodes are HTML pages
 - Edges are hyperlinks
- Each crawling thread know its current location and the content of past pages. It uses the graph to find the path to take to get to the desired resource, requesting as it goes

Paths to Pages vs. Deep Links

- Green and blue are crawling threads
- Blue is at E, Green at A
- Color shows history
- Link Queue: F, D
- Green told to go to F
- Green consults graph, finds path to F which is A-B-E-F
- Green requests B *even though in master view we already have it!*
- Green requests E
- Green requests F



Paths to Pages vs. Deep Links

- Key is to use shared graph solely for finding paths to resources.
- Uses modified BFS algorithm to find paths inside graph
 - Uses random collection instead of BFS's queue or DFS's stack means path is reasonably short, but not always the same path between 2 nodes.
- Since graph is unidirectional, its possible there is no path from a crawling thread's current position to destination
 - But our crawling thread acts just like a browser, including a URL history and forward/back buttons
 - No path from current position, simply “*go back*” one URL in the history and find path from there

Session State Issues

- HTTP is stateless. Web application keep state using:
 - Cookies
 - In URL
- Must ensure that session state associated with the crawling thread that found a new link doesn't get reset by a different crawling thread
- Crawling threads don't share cookies
- Crawling threads do a common pending URL
- In URL session state could hurt us

`s.org/buy.php?item=139&sid=0WS35SHHGEXOPD3WSQP7`

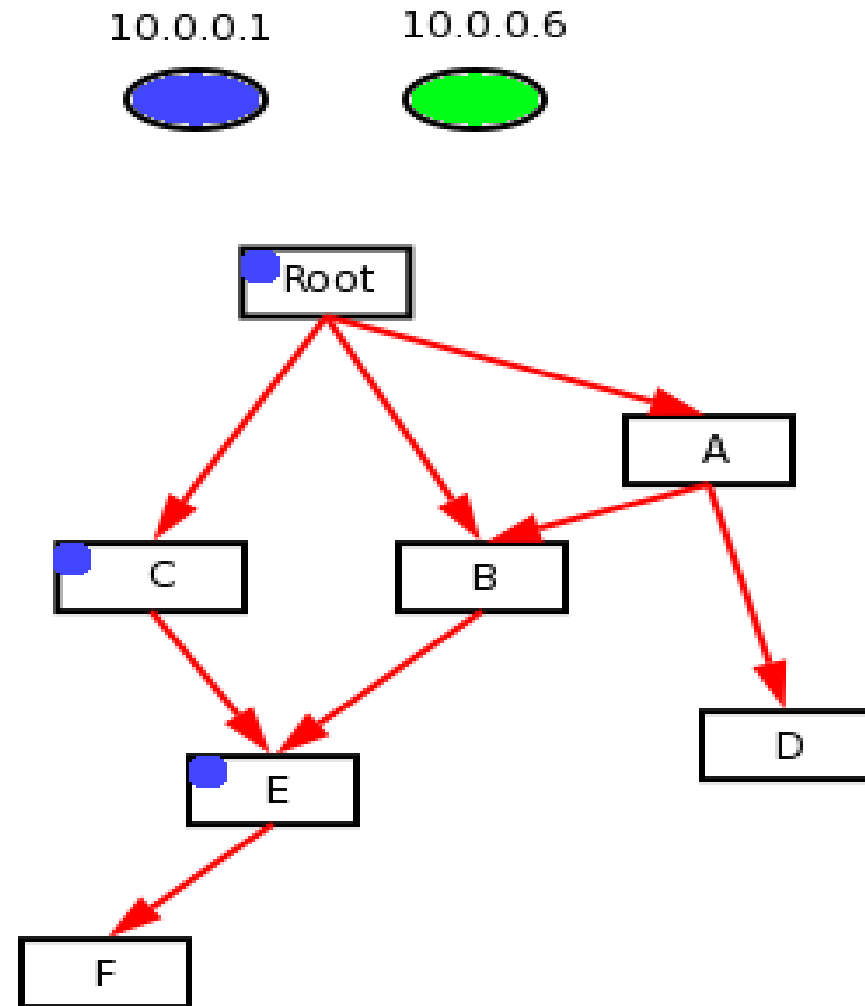
`s.org/buy.php?item=139&sid=7AXOI63NUQ2TTEYFB1CP`

Session State issues

- Again using a graph and paths help us
- Since the crawler never jumps directly to a target page but instead follows a path to that page, we keep our session state along the way!
- This is best shown by example

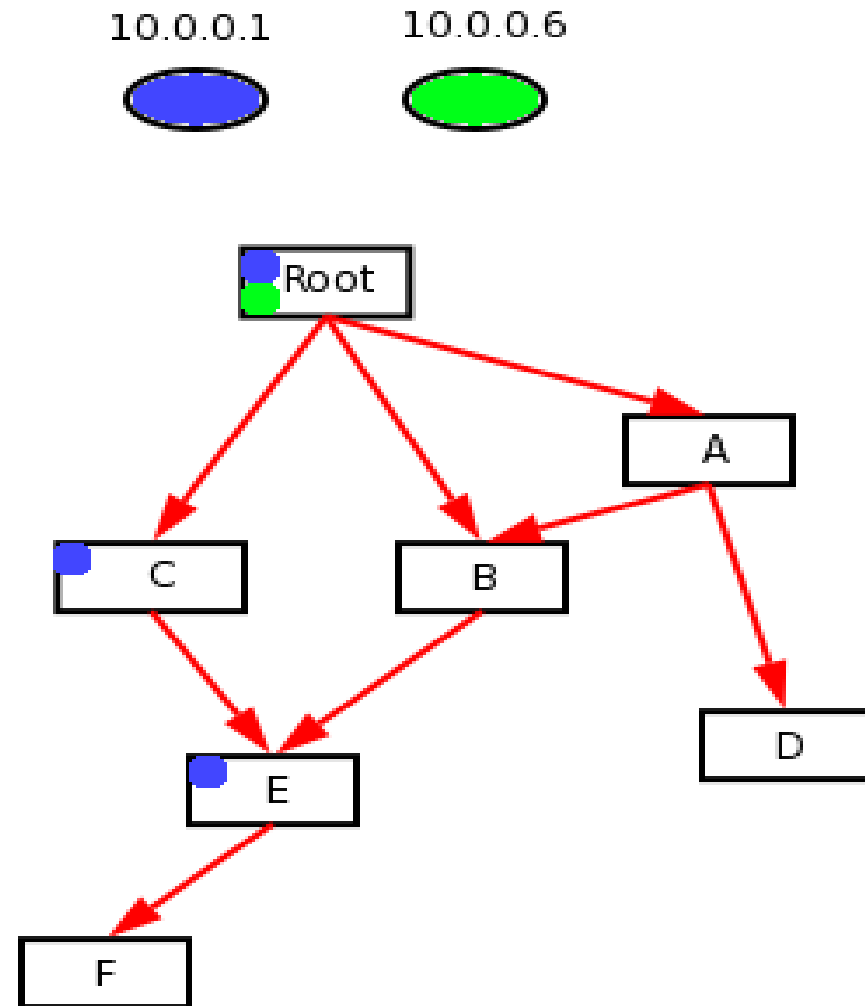
Session State - Frame 1

- Blue has visited root, C, and E
- Link queue is B, F, A
- Graph's nodes are defined by URLs that are "polluted" with Blue's in URL session state (sid=...)
- Green is spawned and told to go to B
- Green uses path Root-B
- Green makes a request for the root (which is not polluted by Blue's session state)



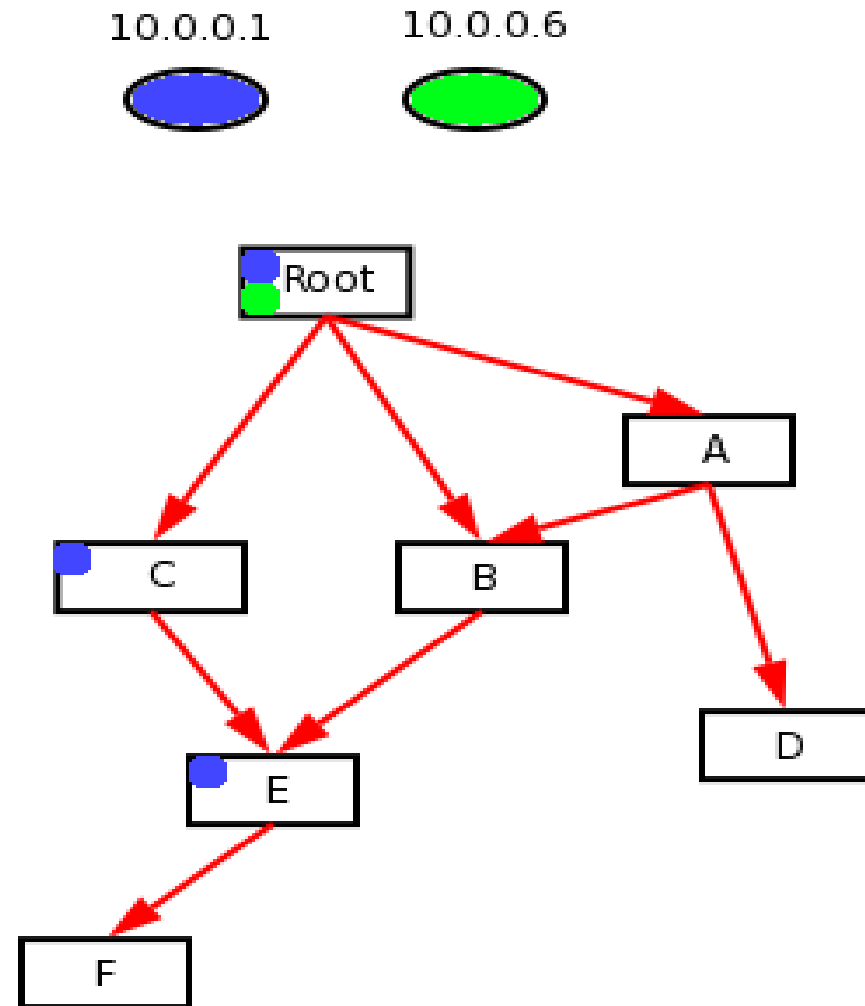
Session State - Frame 2

- Blue is at E, Green is at Root
- Link queue is F, A
- Green makes request for Root, gets it.
- Green knows it will have a link to B (from the shared graph)
- Green knows what link to B looks like (but it contains B's session state)
- Green scans all links on root, looking for link that has the same path/filename but different session state



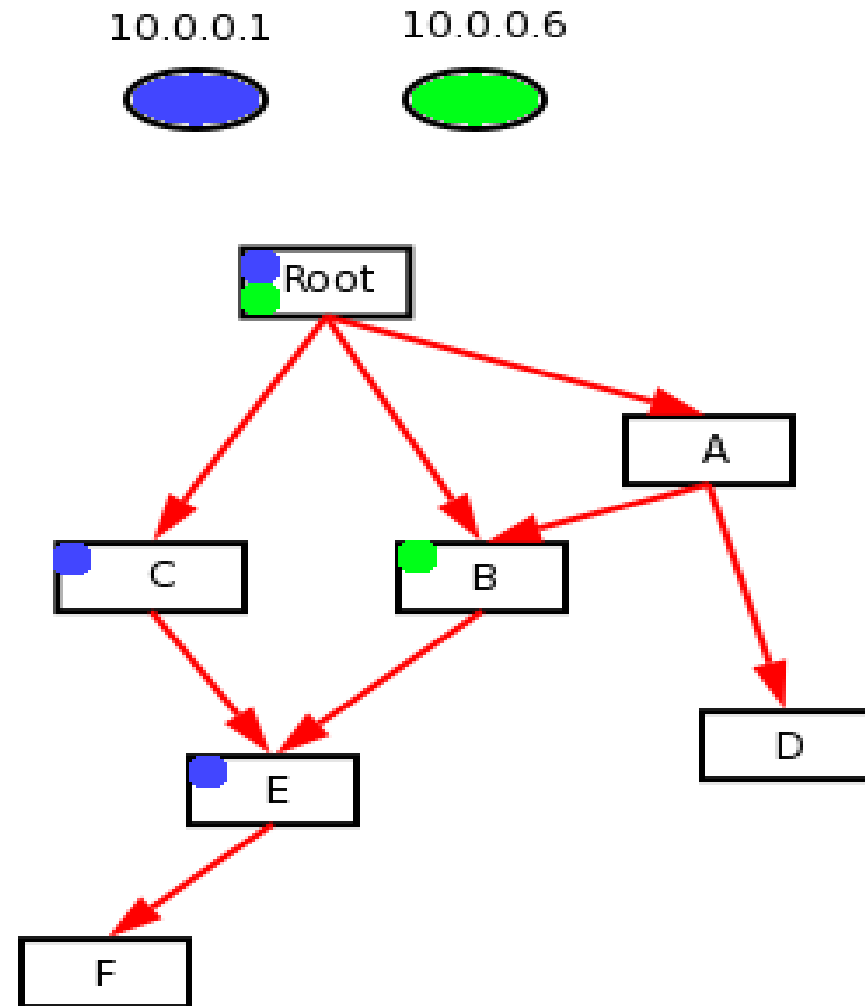
Session State – Frame 2 (Cont'd)

- Blue is at E, Green is at Root
- Link queue is F, A
- Green looks at a diff of the links, finds link to B on its copy of root
- This link will contain Green's session state
- Green can now make a request for B correctly
- Green makes the request for page B



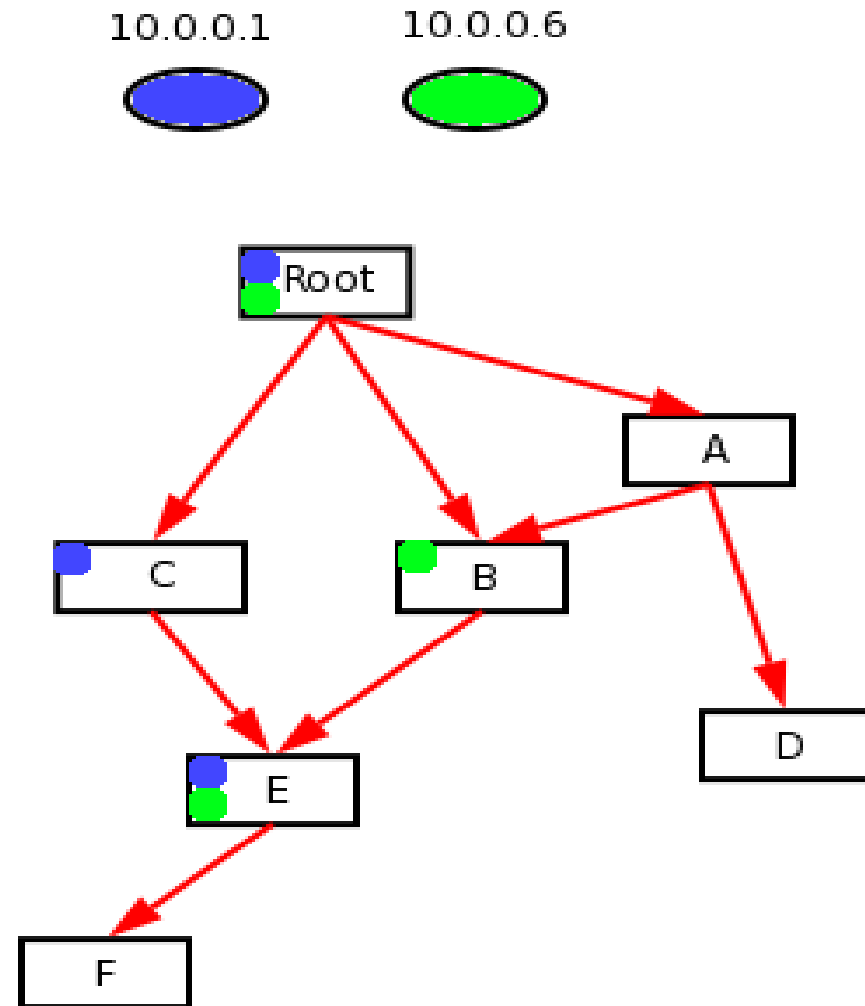
Session State – Frame 3

- Blue is at E, Green is at B
- Link queue is F, A
- Green now told to go to F
- Green looks at graph, finds path B-E-F
- Green looks at its copy of B and using the URL for E in the shared graph (which is “polluted”) finds its version of the link to page E
- Green now makes a request for page E



Session State – Frame 4

- Blue is at E, Green is at E
- Link queue is A
- Green knows its one hop away from its destination, page F
- Green looks at its copy of E and using the URL for F in the shared graph (which is “polluted”) finds its version of the link to page F
- Green now makes a request for page F
- Green has successfully retrieved page F, always with the proper session state



Path to Pages - Conclusions

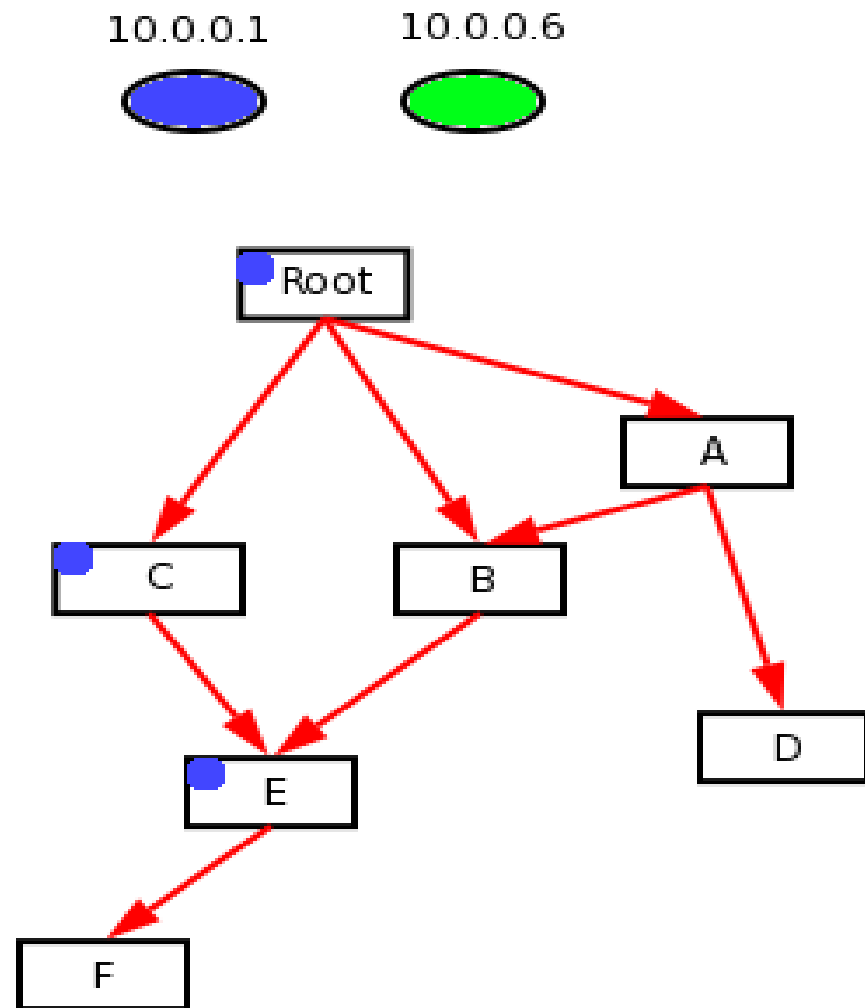
- Any links that Green finds on page F will be added into the pending queue if they don't already exist, and added to the tree
- Links in the pending queue are related to nodes in the graph. All pending links are leaf nodes
- Crawling thread will overlap and will download same pages multiple times
- Only way to make each appear as a separate user on a separate IP

Alternatives to Path To Pages

- Users also can access deep linked pages from a search engine or another page
- Covert crawler should create fake Google referrers and can deep jump directly into a website
- It looks as if we came in from Google
- Google referrer header will contain search terms we used but its all fake.
- Crawler cannot jump to a leaf node.
- If a URL is a leaf node, we have no visited it yet and don't know its content
- Without content, we cannot fake a Google referrer

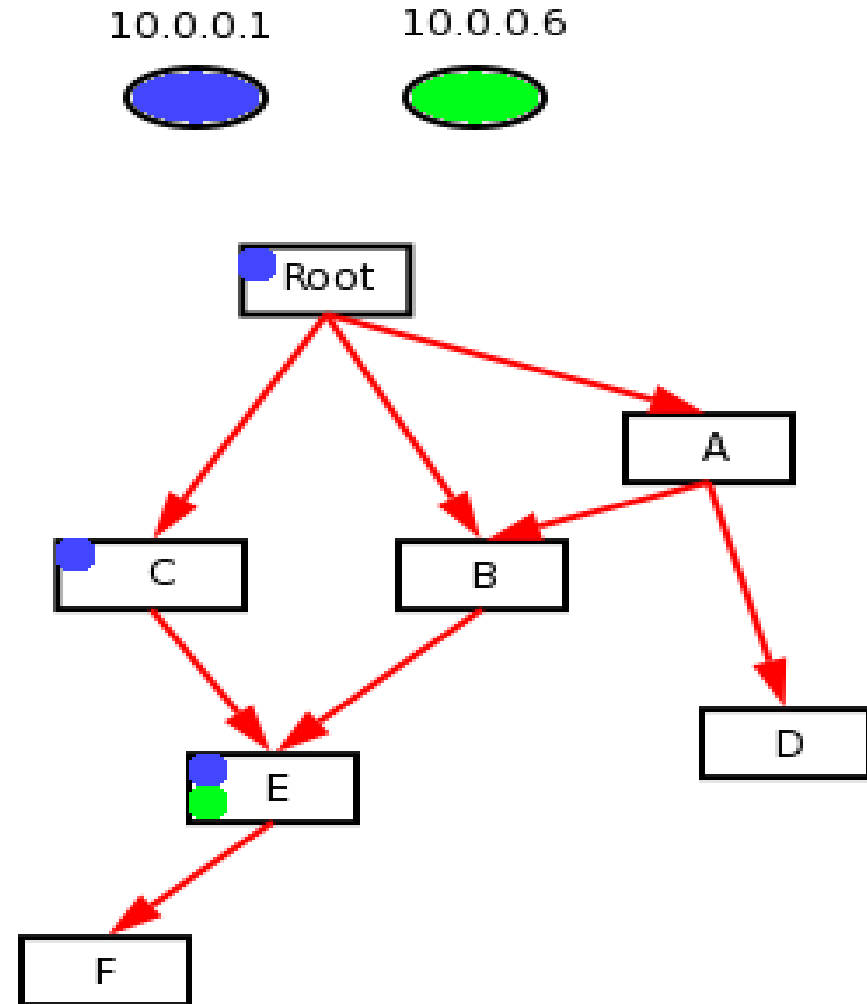
Deep Jumping with Fake HTTP Headers

- Blue is at E, Green is at nowhere
- Link queue is F, B, A
- Green is spawned and told to go to page F
- Decides to do a deep jump
- Found a page E that links to page F
- Green looks at the content of page E for keywords
- Green creates a request for page E with the “Referer” (sic) header set to a Google query (google.com/?q=) for those keywords



Deep Jumping with Fake HTTP Headers

- Blue is at E, Green is at E
- Link queue is B, A
- Green received page E
- Green knows it needs to go to F
- Green looks at its copy of E and using the URL for F in the shared graph (which is “polluted”) finds its version of the link to page F
- Green sends the request for F
- All done!



Throttling Issues

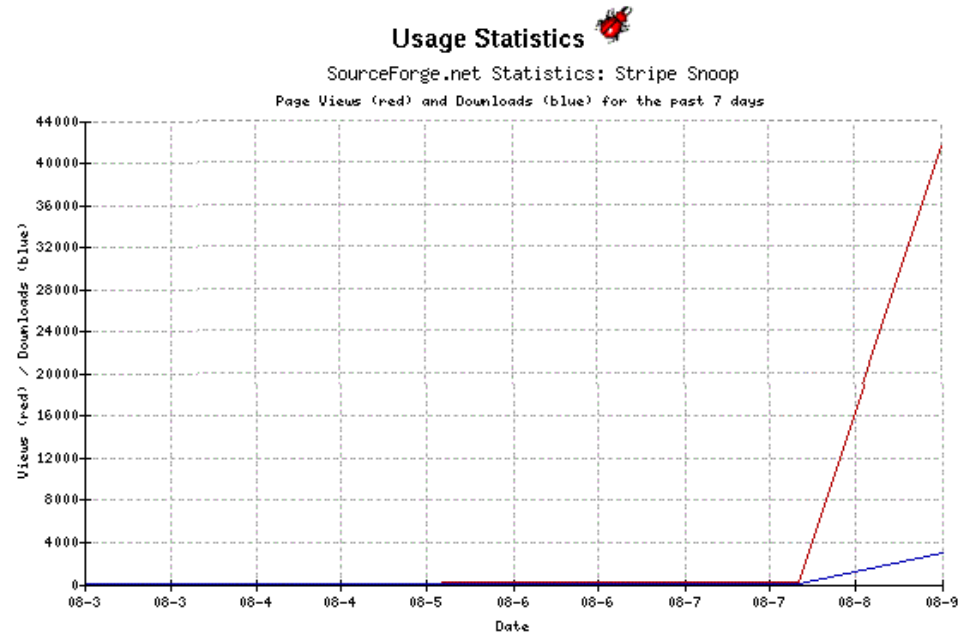
- Need some mechanism to determine how popular a site is to throttle how many crawling threads to have and how often they run
- IP Fragment ID (Fyodor's *How to Own a Continent* chapter)
- WHOIS to find site age
- Archive.org to find how often it's updated
- Google to find number inbound links (popularity)
- Google to find size of site (that Google can index)
- Alexa, other services for popularity info
- Really hard to do! Still refining it.

Traffic Escalation

What is Traffic Escalation

- A site receiving orders of magnitude more traffic than normal
- Big news story, blog
- Slashdot, The Register, CNET, etc

Page Views	D/I
41,552	2,944
185	6
163	26
298	24
67	8



Must You Always Be Covert?

- We've focused on throttling a crawl to match a site's traffic patterns
- If the site is getting flooded *normal traffic patterns* are not longer relevant!
- Traffic escalation allows malicious users to
 - Increase the speed of the crawler
 - Increase the number of pseudo-browser threads crawling the site
 - Increase max number of pages each pseudo-browser can visit
- Covert methods should still be used! More traffic doesn't negate browser emulation or intelligent link selection

Predicting a Flood?

- How do you predict a site that will get massive amount of traffic?
- Ask why traffic escalation happens
- A link appears on a popular site to a relatively less popular site escalating the traffic of the lesser site.
- Major new sites and blogs have RSS feeds...
- Write a simple agent that subscribes to major RSS feeds. Scan new stories for links, checks Google, etc, for referenced sites' popularity. Notify user when less popular site is linked and possibly under heavy assault.

Causing a Traffic Escalation (Slashbombing)

- Sometimes traffic floods need a little nudge
 - Find some interesting content on a site you want to crawl
 - Submit it to a popular news/blog site and see if they pick up the story
- Sometimes traffic floods need a ruptured dam
 - Quickly find a cross site scripting (XSS) vulnerability in a major news site. They are everywhere
 - Exploit XSS to serve a fake inflammatory article (simple document.write, no DB exploitation)
 - Submit link to fake inflammatory story to another major news site
 - Watch the flood

Implementation of a Covert Crawler

Covert Crawler

- Written in Java
- Emulates a Windows XP SP2 IE browser
- Uses scoring to make link request decisions
- Multiple threaded across any number of IP's
 - Uses graph and modified BFS to create random , reasonably short paths to pages
 - Session state issues resolved by not sharing cookies and diffing URLs to avoid in URL state
- Uses “personalities” to determine how long a page is viewed.
- Source code will be released soon. See <http://www.spidynamics.com/spilabs/>

Final Thoughts

Sanity Checks

- Someone does not have to crawl the entire site, just enough to learn the structure and technologies
 - CNN.com is ~220,000 pages
 - But only has a few footholds
 - Stock ticker
 - Story content system
 - Video server
 - Email alerts
 - Search engine
 - A malicious user only needs a few samples of each. More are unnecessary

Summary

- Crawlers can be programmed to send requests just like a browser does
- Crawlers can make intelligent link selections that mimics human behavior (to some extent)
- Covert crawling can be scaled across multiple threads and IPs without revealing there is one mast crawl going on
- Session state can be handled by not sharing cookies and only following paths to pages instead of just requesting of deep links.
- Throttling is tricky, but can be done
- Traffic can be increased to hide any mistakes
- Proxy selection is a weakness in this system but this can be mitigated

What to Take Away From All This

- You cannot rely on your logs
 - To tell you when you are being scoped out as a target
 - To tell you any one user is grabbing large parts of your website
 - To reveal who has been testing you after you discover an attack
- Logs are a passive defense. Stop being passive!
- Fix the vulnerabilities in your web applications

Questions?



Covert Crawling: A Wolf Among Lambs

Billy Hoffman (bhoffman@spidynamics.com)

SPI Labs Security Researcher