21c3 - Automated hacking via Google

By Daniel Bartlett
December 2004
<danbuk@gmail.com>

1. Bio/Intro

Hi, my name is Daniel Bartlett (aka. DanBUK). I have been playing with/on the Internet for the last ten years. I have run a couple of servers, written a few small applications in C, PHP, Perl and unfortunately VB?!

I have been working as a Network Administrator for about three years; yes it's mainly a Windows network. But I'm gradually getting Linux in there.

2. The freebox security and development team

I am a member of The freebox security and development team. A small group of people spread over the world that enjoy discussing, trying, experimenting various random ideas. We enjoy a challenging task because it gives us something to work on and argue over!

We have been involved in a number of open source projects my favourite was/is fb-livecd. A tool for creating custom bootable Gentoo Linux live-cds. Our presence on the net is mainly via Silc/Jabber and email since I now have no time to work on the website.

3. Outline

Today I will be discussing the fun that is possible using PHP. There are a large number of sites that run PHP. Many people start coding in PHP; either their own site or a project/web application. But they do not always thing securely.

The automation of hacking/cracking whichever word you prefer (I prefer cracking in this context.), can be done in many fashions. I started these ideas with a tool written in C for Googling known issues with a project/web application and testing the results. A little lame I know but it got me thinking. Continued that idea to error messages, but rather than having the know variable utilised a set of common variables. Again a quick method, but not advanced.

The most rewarding is manually walking the pages/forms of a site. Very time consuming, but can be automated. I am going to discuss all of these and work towards an automated tool utilising search engines and other people’s servers with the final goal of a PHP-Worm.

4. PHP

PHP is a very easy language to learn, there is a plethora of tutorials and example code to learn from. There are many modules that can be used and classes to aid in the development process. This is a good thing for rapid development. But many people overlook what their code allows to be done in all circumstances. This leads to lots of exploitable code. I'm going to explore this.

4.1. Issues

The main issues are caused by people not sanity checking the variables that are used in the code. This can be rectified by running one function on all variables that will be used in the code. This function should ideally contain a 'white list' of the permutations of what is expected. But when this is not feasible it should remove anything that could cause exploitation. An example function to 'white list' the data is as follows:

```
function white_list($indata) {
      $white = array('home', 'products', 'contact');
      if(in_array($indata, $white)) return $indata;
      else return '';
}
```

An example of a 'black list' function is:

```
function black_list($indata) {
      $black = array('http', 'union', '..');
      for_each($black as $value) {
            $indata = str_replace($value, "", $indata);
      }
      return $indata;
}
```

If this is not done it can lead to one if not all of the following:

Local File Inclusion - Information disclosure, execution of scrips in the incorrect context, execution of uploaded scripts.

Remote File Inclusion - Foreign code execution, obviously the most dangerous. Can lead to lots of things happening on their box.

SQL Injection - Information disclosure, passwords, email addresses, any data that is stored in that database. Sometimes all databases; if that have been silly and used the root user account!

File Upload - Can be utilised in conjunction with local file inclusion. In some cases if the area of upload is within the web root direct execution or overwriting of content.

5. PHP Include Test Script

In testing of the vulnerable scripts I worked over a period of time on an include-able script. With assistance from a fellow freebox member, fukami. We developed a very handy tool for exploring a server.

Browsing directories with writable statuses, viewing and editing pages or PHP scripts, uploading of files to anywhere that is writable, running command line based applications a sudo command line, browsing database servers, running TCP port scans, sending MIME emails, global variable debugging as well as debugging of the script itself.

The script runs on probably 95% of the site I have tested it on. When it fails to run on a

site sometimes I get highly frustrated and spend hours working out why it won't run on that specific installation. The aim is to have functionality that can be run on ANY system.

This script has grown to 45k so far and will probably grow in the future, but it does contain many useful functions that I'll probably utilise in the future worm. I shall now demonstrate a few sites that you might have seen before and how much can be gleaned from these methods and script.

6. Automation

The automation of exploiting is less rewarding that manual attack, but is the most rapid. If you take say a know vulnerability in a version of phpBB and Google the version take the results and tack the exploit on the end with a 'checksum' piece of data you can verify if the site is vulnerable. I coded a small piece of C to do just that. Google, parse the results then test each of the sites. It would run x number of process from the results concurrently so it was high on bandwidth and sort of on CPU time, but quite efficient no files were downloaded other than the HTML. It would take say 60 seconds to run on a 512k ADSL connection for 20 results from google.

The same tool I modified to also work with error codes. Get Google results for 'Failed opening for inclusion' and 'fuzz' them with commonly utilised variables. This generated quite a few false positives. In order to overcome that, I placed a small include file on the web that would echo the variable that called it. This brought the results to nearly 100%.

The next step is to 'walk' peoples sites, trying each pages links and forms. For this I started moving onto PHP. I converted the Google parsing into PHP and built some functions to enumerate the links from the pages for building an array of links for testing. This lead to a simple front end where you provide the search string. It then 'walks' through the steps. Google, grab page then test each with the known variable and the 'fuzz set' using the small test PHP include to verify if vulnerable. Getting approx 90% vulnerable pages. Then just a Ctrl+Click (I'm a firefox addict, USE IT! It beats IE hands down!) and you're at new site to play and explore. In order to be more concealed or if you are paranoid you can of course proxy these actions using a proxy or a translation site.

Then next step is to combine the Google and parsing of links of other sites. This process is not a very difficult one to code, but getting it 100% correct is time consuming. Running these tasks takes time, so the command line execution it the obvious choice, but I want this to be web based so if you set the max_execution time at the beginning of the script. On many servers you can.

7. PHP-Worm

The culmination of the automation of these processes is a PHP-Worm. I researched this idea on the Internet and I only found code for worms that would infect all the local file on a box. I wanted to go one better, well a lot better. I have been aiming for a single script to find other sites, test their pages and infect, and spread by itself. Looking at this task kind of overwhelmed me, so I started with the basics. A function to 'walk' the web root building an array of all files and directories and their writable status. This takes a little while so the data is then stored for future reference. We locate a writable directory within the web root and place the main body of the code in place. Then we start the main infection with a web request to it. We cannot allow this

execution to take too long as the majority of further steps will be caused by someone browsing the web.

In order to do this I worked towards a status system so each step only does a small part. Breaking up the web walking and testing into smaller steps, IE. the first execution gets the results, the second tries link one, etc. On a positive result start or queue the infection. Now that covers the spreading of the worm on a network sense, the local infection is a lot more rapid. Utilising the array of data about the file system build earlier we can quickly modify existing pages to cause more executions of the worm. In a previous project I was working on, I was generating images from PHP so that lead me onto adding a small image into peoples pages that is actually executing the worm. So take all the writable HTML/PHP and find the start or end of the page(<BODY>) and insert the image tag to call the script. Another option I considered was a pop up window, but I quickly dismissed this due to all the pop up blocking that everyone employs nowadays. Also an IFRAME could be an effective option and used for mass messages once the network is built up.

Once the node is nearly all infected I had been considering utilising client side vulnerabilities. Place the exploit in the web pages and cause the client to download another form of the worm. Would need to be binary and OS specific but the testing for the OS can be done from the HTTP headers sent by the client. This could then be called by it's parent to do the time intensive processes that would might be cancelled by the web browsing client. Though this is still a pipe dream.

In the same dream I wanted to work out some form of P2P between all the nodes of the worm. The simple network protocol I had been of went something like this. As an infection happens the parent and child both store a reference to each other. Then once an infection is complete the can start to share information. The information they might share would be along the lines of sibling nodes, possibly to speed up communication via tracing routes and finding nodes that are on route to each other. Task distribution; one gathers the result set from a search and farms out the tests to other nodes. To start executing other nodes; if an infection is just starting out the process it has to run to establish itself are execution/time hungry. So to speed up it growing up, run it a few times. Farming of other activities such as a simple web GET DDos or spam distribution. Which probably wouldn't be black listed due to not being in the dynamic IP range and coming from a valid domain. And of course it should allow an update to the worms' core to be sent out. Especially in the case of someone releasing a Snort filter for the worm. The possibility of building up a front end to control the whole network from any node. In that front end you should be able to send say a command line execution of say 'ls / -R -a' and get the results emailed or ftp'ed to a central location (or the whole network) for further analysis.

A. Further ideas that need development

Mutation - Encryption, Variable replacement, Obfuscation - To aid with hiding from Squid/IDS'es, to limit others modifiying and re-releasing the code.

Multi-Language - In the same essence as the client side binary. ASP/Bash/Perl so we can get into all systems ;)