RZ 3255 (# 93301) 06/26/00
Computer Science 71 pages

Research Report

Cercator: Vulnerabilities Detection by Means of Search Engines

Guy Wald

IBM Research
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

Cercator: Vulnerabilities Detection by Means of Search Engines

Guy Wald

IBM Research, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland

Abstract

Misconfigured machines on the Internet provide information to search engines that can be used to identify them as vulnerable sites. However, most people are unaware that their site is exposed. Using simple queries, a hacker is able to locate vulnerable sites by searching for the "signatures" of typical vulnerabilities.
The goal of this project is to evaluate in how far it is possible to gain sensible information about remote sites by only questioning search engines and without ever contacting the site. Afterwards we will develop a strategy that allows to search for vulnerable sites in a systematic and automatic way. As that strategy needs a special tool to contact several search engines at once, we will develop that as well. The tool will be very general and will allow to make efficiently very complex searches about any topic. At the end a series of tests allow us to conclude about the usability of the method.

Contents

Chapter 1

Introduction

Currently we notice a trend claiming for more security in computer networks after the distributed denial of service (DDOS) attacks earlier this year or after the recent ILOVEYOU virus. This project situates quite well in this context. It presents a new way to discover vulnerable sites, that is to say computers that are likely to being hacked.

The new idea here is that we want to analyse, if it is possible to get sensible information about remote sites in a different way as it is done up to now. We do not want do enter in direct contact with a remote site, but we want to gain the information in an indirect way, by using search engines. The idea is not so new, because Black Watch Labs published an article about that subject and its potential dangers in February 2000 [1, 2]. This original approach has two advantages for a potentially malicious hacker. First it allows him to get immediately information about the whole Internet without contacting each site individually. And the second advantage is, that he gets the information about remote sites without leaving a trace in any log files there. It is important to underline that by only submitting queries to search engines, we do not actually exploit a vulnerability.

In this project, we will develop a new way of searching. To do so, we will analyse the search engines and evaluate their performances. We will look particularly carefully at the ways there are to formulate precise queries. Once we have all this preliminary knowledge, we can elaborate the strategy. We will build the tools necessary to apply the method in a convenient way. At the end, we will use that new general searching tool for a particular case: we will use it to search for security holes. We will try to find out if search engines can deliver critical information about vulnerable sites and whether we can use the tool in a precautionary way in order to warn sites that their computers seem to be misconfigured and therefore perhaps vulnerable.

Chapter 2

Background

2.1 Vulnerabilities detection

2.1.1 Vulnerabilities

In this context we call vulnerabilities, bugs that can offer control over a host to everybody who knows how to exploit them. We can distinguish two main sources of vulnerability: bugs in software or bad configuration of correct software. The latter one is more frequent. For example a config file for the web server, that is set up badly, could make the whole file system readable from the world wide web, and that would not be very healthy for the concerned site.

When simply speaking, we can say, that security bugs are due to a too optimistic design or coding either of the software or of the protocol in question. The designers of the vulnerable software or protocol spend so much effort on an efficient specification related to the main function of the product, that they forget to consider special cases. The designers do not take the time to imagine where potential problems could arise. Therefore the product will achieve its primary purpose very well, but the bugs are due to unspecified states.

A good design however is not yet a guarantee for a good program, because programmers too are sometimes responsible for errors in programs. The latter ones often are too optimistic when coding. The most frequent bug in vulnerable software is due to a buffer overflow. This error is due to the fact, that programmers did not respect the rule "Never trust user data" and did not check the parameters and variables they used in their code. A buffer overflow can simply crash a software but sometimes a hacker can also succeed to write malicious code in parts of the memory he should not have access to, and then he can execute it. The only solution for having a secure system is to keep ones software up to date all the time and to configure it correctly. In chapter 8 we will apply our new method to find examples for these both types of vulnerabilities.

2.1.2 Classical ways of hacking

It is not easy to give a systematic way of attacking a remote computer system, because each case is different. But there are a few general principles. When a hacker knows which host he wants to attack, he has to find out its network address. That is not a problem. He can use nslookup for example. Then he makes a port-scan on the remote host to see which ports are open. Depending on the way he does it, the port-scan may be easy or nearly impossible to detect in real time in the log files of the remote system. When the open ports are known, the next step is to find out which servers are listening on those ports. Here the degree of difficulty begins to rise. But there are special scanner programs, that can recognise the most frequently used servers. Now that the server is known, there is no more a general rule for continuing. The intruder can try to exploit a

well known bug for one of the installed servers, when, by chance, there is one.

2.2 Search engines

Today one common way to find the information one is looking for, is to use a search engine. That is why there is plenty of documentation about them around [3, 4, 5, 6, 7, 8] with lists of different search engines and with tips to make efficient searches or with explanations about how and how well search engines work. Search engines have a large database of web pages. There are different ways to build and update such a database as we will see in the following paragraphs. A user can access the knowledge of this database by asking the search engine what he is looking for. Then the search engine software tries to understand the user's query, and retrieves all matches from its database. These are the results for the user.

We can distinguish the following categories of search engines:

- Robots based search engines
  - Full text search engines
  - Directory based search engines
  - Specialised search engines
  - Private search engines
- Metacrawlers

The differences between those search engines lie in the way they fill up their database, they interpret the user's query and in which order they present the results. The following paragraphs will analyse in more detail how the search engines in each of these categories work.

2.2.1 Robots based search engines

Some search engines have an agent-program – often called a robot – that traverses the world wide web by crawling from link to link analysing the documents encountered and storing them in a database. These web-walkers or spiders have the advantage that they are very thorough, and can potentially visit all browseable parts of the web. But they have problems too as described in reference [9] and as we will see now.

In the first place, spiders have some bad characteristics . As they retrieve many documents, they generate considerable network overhead, especially when several robots operate simultaneously. They can request documents in such quick succession that the server becomes overloaded.

Another problem is the actual processing of documents. Because spiders come across plenty of documents it is very hard, if not impossible for them, to get a sensible context for an information item. However the most important problem is that they retrieve all documents that can be reached, even those that are not interesting or suitable to index at first glance. But later on, we will show, that in our new strategy, we are relying mostly on that feature.

A way to prevent the spiders from indexing too many files is the use of the robots.txt file defined in the Standard for Robots Exclusion [10]. It consists of a file called robots.txt lying in the top directory of the web server and specifying which robots can index which files. Nowadays nearly all robots follow that standard.

There is another strategy for building a database. It consists of asking users to register their web pages manually at the search engine. That guarantees that the database contains only valid and relevant data. Often the two strategies presented here are used in a combined way.

The problem with nearly all search engines is not to build the database but to keep it up to date. The world wide web pages are changing so fast, that it becomes difficult for the search engines to keep their database in a correct state. Often it arrives that the search engine returns a link to a web page, that does not exist any more. The link is broken. In general the search engines need about three months to verify their whole database content. Later we will see, that for our project, broken links too can be useful, because they tell us, that a certain page once existed. That means, that the software can still exist, but the corresponding web page is no more available on the world wide web.

Full text search engines

Full text search engines use robots to retrieve the web pages. There characteristic lies in the way, they analyse the pages and build up the database. Full text search engines index each word (apart from those little words like: the, and, or, ...) and store the webpages in such a way in their database, that it becomes possible for the user, to find any page only by specifying one or two words that it contains. AllTheWeb and AltaVista are two examples of Full text based search engines.

Directory based search engines

For this type of search engine, each new page is classified into a category corresponding to its content before it is added to the database. Often these categories also have sub categories so that the database gets a structure like a tree. The construction of the database is made either manually or by data mining techniques. When searching some information, the user just has to browse through these categories and sub-categories to find a list of matching sites. Two famous examples for this category are Yahoo and Magellan.

Specialised search engines

Often it is possible to get better results when using specialised search engines that only index pages from a special domain of interest. For example there is a search engine at http://www.mp3.com that only contains MP3 information. There are other search engines that are specialised on Usenet content (e.g. http://www.deja.com). Reference [11] contains a large list of specialised search engines.

Private search engines

These search engines work in the same way as other search engines, but their content is limited to the intranet of one company. They may use the same software as the international search engine. As their database is not so large, it becomes possible for them to verify and update the content more often than the big international search engines can.

2.2.2 Metacrawlers

Metacrawlers are not real search engines as defined above because they do not have neither their own database nor their own robot crawling around. They offer an interface where the user specifies his request and then they rely the request on to other real (robots-based) search engines. The advantages for the user are: he has to specify only once what he is looking for and he gets all the best results from different search engines. Examples: metacrawler.com, SavvySearch.com, ...

2.2.3 Results ranking

Another major point of distinction between different search engines is the way they rank the results. The ranking is something particularly important as the user rarely checks more than the twenty first results he gets back from the search engine. So the search engine software should try to understand from a few keywords what the user is interested in and put the most meaningful results first.

Sometimes search engines use the meta keywords contained in the html file for the ranking. When a lot of meta keywords match the keywords specified by the user, then the result gets a high ranking. In other cases the search engine software analyses how many specified keywords there are in the result file and the proximity of those terms within a page. The more often the keywords are there, the better.

A completely new ranking strategy has been recently developed by Google and encounters a lot of success. It consists of analysing the hyper-links contained in the documents retrieved by the robot. A web site's rank will then depend on the number of other web-sites that are linking to it. So it ranks relevant web-sites based on the link structure of the Internet itself. In essence, Google interprets a link from page A to page B as a vote, by page A, for page B. Google assesses a page's importance by the votes it receives. But Google looks at more than sheer volume of votes or links. It also analyses the page that casts the vote. Votes cast by pages that are themselves "important" weigh more heavily and help to make other pages "important". These important, high-quality results receive a higher rank and will be ordered higher in results.

In general however, search engines do not publish their exact ranking strategy. Otherwise it would become possible for web masters to cheat and they could artificially increase the ranking of their own pages [12]. One exception is presented in [13]. The author has succeeded to crack the AltaVista ranking mechanism in 1997. Since then, it has however changed. It also arrives, that one search engine product is known under different names. For example HotBot, Yahoo, Snap, GoTo and MSN use the search engine software from Inktomi, but with slightly different parameters. That explains why they generally return the same results.

2.2.4 Query syntax

Now let us look into more detail how searches can be specified with the different search engines and which syntax they support for the queries.

Basic query features

There are two main ways for formulating a query. Some search engines understand *boolean phrases* composed of the keywords and of the expressions for a boolean *and*, *or* and *not*. For example a search for everything about the Zurich IBM Research Laboratory could result in the following query: *IBM and Research and Zurich and not (Watson or Almaden)*.

The other way to specify a query is to list some keywords and to add a + sign before those words, that must be included in the results, and to add a − sign when the word must not be among the results. The previous example would look like this: *+IBM +Research +Zurich -Watson -Almaden*. In general it is not possible to transpose a boolean phrase into this format without loosing some information, because the format does not support parenthesis. Compare *(not A or B and (C or not D)* with *-A B C -D* which is not quite the same (*not* has a higher priority than *and* which has a higher priority than *or*).

Another common feature supported by nearly all search engines is *phrase search*. Normally, when only individual keywords are specified, then they do not have to be next to each other. But to guarantee that the keywords have to follow each other, they are put between quotes and then they are called a phrase. Consider the following example: when specifying *International Business*

Machines you will find a lot of pages that contain these three words. The ranking mechanism of the search engine will try to show first the pages, that include the three words written next to each other. On the other side, when specifying *"International Business Machines"* there will be less results than before and it is sure, that the three words appear together.

Advanced query features

Nearly all search engines checked during this project allow to look for phrases and to impose the presence or absence of some keywords in the document. There are also some search engines that offer supplemental features and therefore permit more precise queries. The most frequent specialities encountered allow to limit the search on

- Language
- URL
- Title
- Domain (top level domain)
- Dates
- Link.

2.2.5 Analysis of several common search engines

Full text search engines

Table 2.1 shows the features supported by various commonly used search engines. For this project, search engines with similar features are grouped together in a class. Unfortunately it is not so easy to classify them, as there are always some exceptions. This regrouping will serve us later when searching for vulnerabilities in different ways. Often there is also a national version of the big international search engines, but they have not been considered in this project, as it would not have changed the conclusions. We preferred to concentrate on the global search engines, as in general they have a larger database.

Metacrawlers on the net

The following list shows a certain number of metacrawlers available on the net and a comment about their features.

http://www.metacrawler.com	choose among 12 search engines, restrict domain
http://www.infind.com	calls out in parallel 6 search engines, merges the results, removes redundancies, and clusters the results into neat understandable groupings; does not support special features.
http://www.savvysearch.com	contacts 10 search engines
http://www.mamma.com	In advanced mode allows to choose among 8 search engines and to set the time-outs
http://www.highway61.com	supports 5 search engines allows to specify the time-out, the number of links to retrieve and can check for broken links
http://www.profusion.com	supports 9 search engines allows to specify the number of links to retrieve and can check for broken links
http://www.thebighub.com	supports 8 search engines, allows to specify the time-out and the way the results are presented
http://www.c4.com	supports 12 web search engines and a lot of specialised ones, specify number of hits to retrieve and time-out

Class	Name	AND	OR	phrase	NOT	Title	Domain	URL	Lang	Link	Date	Comment
3	AllTheWeb	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	fast
4	AltaVista	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	other tags: near, host, image, applet, anchor
2	EuroSeek	✓	✓	✓	✓	–	✓	–	✓	–	–	host
2	Excite	✓	✓	✓	✓	–	✓	–	✓	–	–	
1	Galaxy	✓	✓	–	✓	✓	–	–	–	–	–	
1	Google	✓	–	✓	✓	–	–	–	✓	–	–	fast, keeps all pages in a cache
1	GoTo	✓	✓	✓	✓	–	–	–	–	–	–	
4	HotBot	✓	✓	✓	✓	✓	✓	–	✓	✓	✓	
3	Infoseek	✓	✓	✓	✓	✓	✓	✓	–	✓	–	
1	Lycos	✓	✓	✓	✓	–	–	–	–	–	–	
4	NorthernLight	✓	✓	✓	✓	✓	✓	✓	✓	–	✓	slow
3	Raging	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	fast, very recent, other tags: host, image, applet, anchor; powered by AltaVista
1	Yahoo directory based	✓	✓	✓	✓	✓	–	✓	✓	–	–	

Class 1: *AND*, *OR*, *NOT*, *phrases*
Galaxy (exception: no *phrases*), Google (exception: no *or*), GoTo, Lycos
Class 2: Class 1 + *Title* + *Domain*
EuroSeek (exception: no *Title*), Excite (exception: no *Title*), Yahoo (exception: no *Domain*)
Class 3: Class 2 + *URL* + *Language* + *Link*
AllTheWeb, Raging, Infoseek (exception: no *Language*)
Class 4: Class 3 + *Dates*
AltaVista, HotBot (exception: no *URL*), NorthernLight (exception: no *Link*)

Table 2.1: Summary of the main search engines' features

A thorough study of these metacrawlers showed that all of them just take the query composed of keywords or of a phrase and deliver it as is to the search engine. The user has no precise control whether its keywords are implicitly linked by logical *and* or *or*. Furthermore they do not support the special features of each search engine (cf. section 2.2.4), and that is very important for this project.

So it turned out, that none of the major metacrawlers present on the net can be used for this project.

The Copernic metacrawler

By the end of the project, we discovered Copernic[1]. That company develops a metacrawler and sells it in three versions: Copernic 2000 Free, Copernic 2000 Plus and Copernic 2000 Pro. Comparing to the metacrawlers described in section 2.2.5 this program is far more powerful and user friendly. It supports up to 520 search engines divided into 55 categories. It has the possibility to download up to 300 results per search engine and up to 1000 hits per query. Results can be managed into different folders and a report is generated to summarise the results of a query. The program works under MS-Windows and has a convenient user interface, but it is also possible to make it work in a batch-style mode. Among all the results retrieved duplicates are removed. Furthermore a check for broken links is performed and then the results are presented in a convenient manner.

But this program does not seem to support the required features presented in section 2.2.4. So it is not sure, whether it could have been used for this project.

[1] http://www.copernic.com

Chapter 3

The new way of searching

3.1 Basic Idea

The purpose of the project is to evaluate how far it is possible to get pertinent information about remote sites without actually accessing them. The only tool allowed to use are search engines. Now we will present the strategy that we developed to solve this problem.

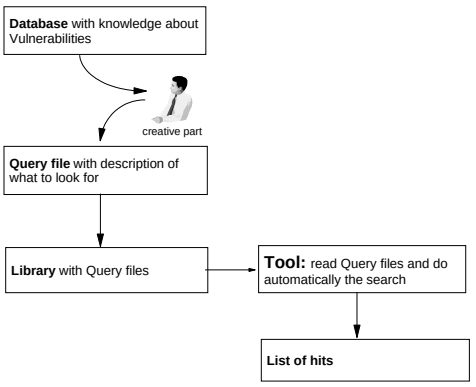


Figure 3.1: Schematic representation of the new strategy

The strategy aims to look in a systematic way for vulnerable sites by using the search engines. First we need a certain knowledge about vulnerabilities and security bugs. That can be kept in a database. Then comes the creative and challenging part of the process: somebody browses through the vulnerabilities database until he finds a topic, where he thinks, that search engines can be used to find sites potentially vulnerable to that bug. He imagines what he has to look for to find out whether a site is affected by a certain bug. Then he formulates some clever queries. This task needs a good knowledge of the functionality of the search engines but also a little imagination. Often it is necessary to make some assumptions before it is possible to start. But of course these hypothesis have to be kept in mind, when interpreting the final results.

Sometimes a simple query searching for the signature of a vulnerable program might not be sufficient to find only relevant sites. It would be desirable to do a second level refinement as the results are not precise enough. That could consist in applying a filter to the results retrieved from the search engines.

All this process is elaborated step by step and by trying out the queries manually. When the person finally has arrived at a point, where its process seems to work and deliver pertinent results, he codes all his queries and filters in a special format and stores them in a file.

Once there is a certain collection of such description files, one can launch a special tool that will take those files as input and do automatically the task of contacting the search engines, retrieving and filtering the results. At the end, the tool will deliver a list with the results returned by the search engines. Then again, this list has to be interpreted by a physical person that decides, whether a site is really vulnerable or whether it is only suspicious or whether the result returned by the search engine is out of context. Figure 3.1 illustrates the whole strategy.

3.2 Initial requirements for the new tool

The strategy just exposed needs a tool, that has to read a file containing the description of the queries to do and then, it has to do automatically the search. As our tests of the metacrawlers in section 2.2.5 showed, that none of them offers the necessary features, we have to create in this project a new metacrawler to fulfil our needs. We called that new metacrawler Cercator [1]. At the beginning of the conception of this new search tool, there were the following requirements.

- Each query should be specified only one time. Cercator will use that input at every execution and for every search engine. So the query has to be specified in a meta format, that allows to formulate queries with any complexity. Cercator then has to understand that format and translate it into a search engine specific format.
- As search engines change quite often the formats of their in- and output and as they offer new features, Cercator should be flexible enough to adapt and take those changes into account. It should also be possible to add easily new features or concepts.
- We decided in paragraph 2.2.5 to group the search engines in different classes according to their features. Now we want to allow different search strategies depending on the class of search engine being used. Cercator should also be able to deal with that requirement.
- A searching strategy may need more than one query.
- As the results returned by search engines are not always precise enough, they have to pass a second level refinement. This feature should be as powerful as possible and eventually allow to replace some feature lacks of the search engines (url, title, ...).
- This filtering feature is closely linked to the ranking idea. With a ranking field, we can differentiate the retrieved hits. When we think, that there is a high chance, that a hit verifying several conditions is good, we do increase its rank and it will be presented before other hits in the results.
- Cercator should use all possibilities that the individual search engines offer to formulate a query and it should retrieve each time a maximum of hits.
- When the program runs in batch mode, it should be possible to specify precisely what to search and where to search. So it has to support several options at the moment of execution.

Those are the initial requirements that we have in mind when designing the Cercator program.

3.3 Implementation choices

3.3.1 Perl 5

When the requirements are clear, we have to choose the best adapted environment and programming language to do the implementation. The program has to send and receive a lot of http

[1] cercare (Italian) means to search

datagrams, but it is too hard to do that in C as C offers only a socket interface. So we would have to add all the upper protocol layers up to the http layer. As other languages – interpreted ones – offer a more convenient interface, we drop C. At that point, it becomes clear, that the program will be a script program.

The program will have to parse the html document received from the search engines in order to isolate the hits (title, urls, descriptions). Such a string treatment works best in a language with a strong support for regular expressions. So Java sorts out.

A closer look at Python and Perl, two script languages that meet the previous requirements, shows, that Perl is by far more widespread and there are packages available for Perl, that can automatically contact search engines, retrieve results and parse them (WWW::SEARCH). That feature determines our choice for using Perl.

Perl is available for most operating systems. We choose to work under Unix (AIX) because it is easier to compile new modules under Unix than under Windows and because Unix has a better support for inter process communication (IPC) and multitasking than Windows.

3.3.2 XML syntax

The input file is the most valuable piece in the whole strategy because it contains the knowledge about vulnerabilities in the system. Therefore it is important to design its format very thoroughly and, by that, to offer a maximum of flexibility. It should have a simple syntax that allows to create quickly an input file. For the programmer of the tool, it will also be easier to write the parser, when the syntax is easy.

The eXtensible Mark-up Language (XML) is a meta-language for defining mark-up languages [14]. In contrast to other mark-up languages, the XML users can define their own tag sets to describe the document content. XML is becoming very popular and it is used in many situations. A closer lookup reveals that it can be used for our purpose [15].

XML has a normalised syntax and it is easy to learn and use. There are packages available in Perl to deal with XML documents. These packages parse the document and build the Document Object Model (DOM). The DOM is a normalised data structure that represents the XML document in a tree-like fashion. The parser automatically checks if the input file is well-formed, otherwise it would not be possible to parse it. In XML terminology, a document is said well-formed, when all tags are correctly opened and closed and when tags do not overlap. It is also possible to check whether the tags are used in the right order. In that case, the XML document is checked against its Document Type Definition (DTD). If that check is all right, the document is said to be valid.

Chapter 4

The Input file

4.1 Description

We conceived a highly hierarchical structure for that file. It includes many functions in order to offer a maximum of flexibility to the creator of the input files. Figure 4.1 shows the structure. One input file can contain the descriptions for more than one subject. From now on, we will call that part an attack. It is a description composed of queries and filters that aims to find out if the remote site has a certain vulnerability and is therefore likely to be attacked.

As described in section 2.2.5, we divide search engines into several categories depending on the features they support. By that, we can allow different strategies to find results in the same attack part. For example if a search engine does not support any special features, then the query part will look only for general keywords, but it may be followed by a filter that checks the title of the document. If we use a more powerful search engine, we can directly search for a document with a special title and do not need a filter behind. This distinction is done in the part called class in the figure 4.1.

Once the class of the search engines that we want to consider is fixed, we can begin with a query bloc. The query bloc contains the specifications of what to search. A query may be followed by a filter bloc, if the creator judges, that he needs to refine in more details the results received from the search engines. In a class section, the number of queries is not limited. It may arrive, that for one attack, there are a lot of different ways to search. Once all queries and their respective filters have been executed, it may be useful to do again a filter, but over all the results together. That will also be possible.

Once all classes are treated, the results are put together. Then it might again be interesting to have a filter e.g. to control whether all links are still valid or broken. Then all the results found during the processing are displayed in a list.

4.2 XML format of the input file

4.2.1 Overview

Listing 4.1 shows how that hierarchy of the input file is coded in XML. Each XML document begins by declaring which version of XML is used. If the underlying document conforms to a DTD, that is specified as well. After these typical XML specifications, the content of the file begins. The top level tag is called <wanted>. The second tag has to be <attack>. Inside the <attack> bloc, lies the whole description of the attack with queries and filters. A file may contain several <attack> blocs.

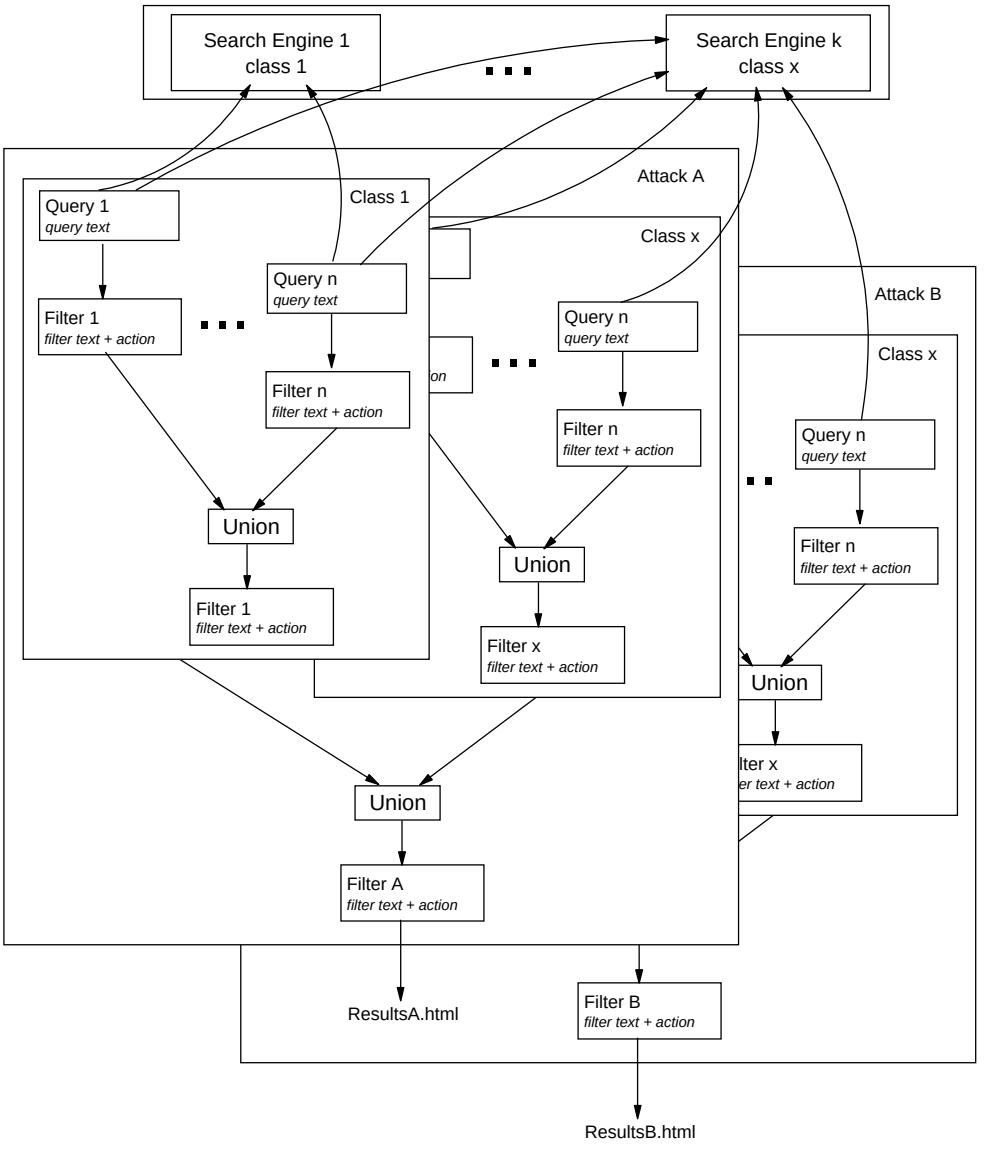


Figure 4.1: Plan of the input file

```
<?xml version="1.0"?>
<!DOCTYPE wanted SYSTEM "wanted.dtd">

<wanted>
   <attack>
      <header>...</header>
      <ref>
         <CVE> ... </CVE>
         <BID> ... </BID>
      </ref>

    <!-- this is a XML comment -->

      <body>
         <seclass param="1">
            <query>
            <phi>

            <query>
         </seclass>

         <seclass param="2">
            <query>
         </seclass>
         <phi>
      </body>
       <phi>
   </attack>

   <attack>
      ....
   </attack>
</wanted>
```

Listing 4.1: Overall structure of an Input File

The next bloc must be a <header> bloc with a description of the attack. The <ref> bloc is optional and can contain a reference number (CVE [16] or BID [17]) of the underlying vulnerability. That reference facilitates to find the description file of the attack that served as inspiration for current the XML file.

The <body> bloc contains all <seclass> blocs. The <seclass> tag gets as parameter the class of the search engine to use. There may be as many <seclass> blocs as search engine classes have been defined. Inside the <seclass> bloc comes the alternation of <query> blocs followed eventually by a <phi>[1] bloc.

In XML comments can be put anywhere in the file and have to be enclosed in <!-- --> tags. In the following sections we will see, that almost each level in this hierarchy is treated by a special module in the program.

4.2.2 Query bloc

```
<query>
   <and>
      <word>Apache User's Guide</word>
      <title>documentation</title>
      <not>
         <or>
            <url>.htm</url>
            <url>.txt</url>
         </or>
      </not>
   </and>
</query>
```

Listing 4.2: Structure of a Query Bloc

Listing 4.2 represents an example of a query bloc. It specifies how the different keywords are bound together logically. The logical structure is given by the three tags <and>, <or> and <not>. The allowed tags that characterise the keywords are: <word>, <domain>, <title>, <url>

[1] phi means filter

<host>, <link>, <fromdate> and <todate>. This structure allows to build all queries with any complexity. But according to the class the query belongs to, not all keywords are allowed.

Cercator will transpose the XML query in an intermediate format which resembles the format that one uses when contacting the search engine manually via their graphical user interface on the web. For the example from listing 4.2 that meta format would look like: *"Apache User's Guide" and title:documentation and not (url:.htm or url:.txt)*. This format will the be transposed in a search engine specific form.

4.2.3 Phi bloc

```
<filter>
   <rule applyto="summary">
      <condition applyto="title">
         $x=~count.cgi
      </condition>
      <then>
         <throw/>
      </then>
      <else>
         <rule applyto="file">
            <condition applyto="all">
               $x=~/root::/
            </condition>
            <then>
               <rank value="50"/>
            </then>
         </rule>
      </else>
   </rule>
   <rule>
   ...
   </rule>
</filter>
```

Listing 4.3: Structure of a Phi Bloc

The <phi> bloc contains the description of the filter (cf. Listing 4.3). A filter allows to refine the results returned by the search engines. A refinement is useful to get more pertinent results or to replace the lack of feature of certain search engines.

A <phi> bloc is composed of one or several rules. A <rule> contains the active part of the filter. It has to contain a <condition> and one or two actions (<then> and <else>). When the <phi> part is to be executed, then Cercator evaluates the <condition> for each hit. The result of that condition is boolean and determines which action to execute.

A rule applies by default to the *summary* of the hit, that means to all the information returned by the search engine. But it is also possible to apply it on the *file* referenced by the url.

A <condition> applies either to the whole *summary* or to some specific part of it. That part is stored in $x and is used in the condition. The condition is a Perl expression that will be evaluated and has to return true or false.

The actions to take are specified in the <then> or <else> bloc. A <then> bloc is mandatory whereas the <else> bloc is optional. Three actions are supported: <rule>, <rank> or <throw>. <rule> begins a new rule bloc as before and can be used to specify a second filter. With <rank> one can modify the ranking of the current hit. <throw> throws away the current hit. That means, that when a hit does not meet the condition and when there is no else bloc, it is not thrown away, but it will be kept unchanged within the list.

After all these descriptions, the structure and the syntax might seem quite complicated and it seems difficult to write those files correctly by hand. That is why, in section 7.1 we will present a graphical user interface that helps writing those files quickly and in a correct XML.

4.3 DTD

```
<!ELEMENT wanted (attack+)>
<!ELEMENT attack (header, ref?, body, phi?)>
<!ELEMENT header (#PCDATA)>
<!ELEMENT ref ((BID|CVE)|(BID & CVE))>
<!ELEMENT BID (#PCDATA)>
<!ELEMENT CVE (#PCDATA)>
<!ELEMENT body ((seclass, phi?)+)>
<!ELEMENT seclass ((query, phi?)+)>
<!ATTLIST seclass param (1|2|3|4) #REQUIRED>
<!ELEMENT query (and|or|not|word)>
<!ELEMENT and (or|not|lang|fromdate|todate|domain|host|word|title|url|link)+>
<!ELEMENT or (and|not|lang|fromdate|todate|domain|host|word|title|url|link)+>
<!ELEMENT not (or|and|word|title|url|link)+>
<!ELEMENT lang (#PCDATA)>
<!ELEMENT fromdate (#PCDATA)>
<!ELEMENT todate (#PCDATA)>
<!ELEMENT domain (#PCDATA)>
<!ELEMENT host (#PCDATA)>
<!ELEMENT word (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT url (#PCDATA)>
<!ELEMENT link (#PCDATA)>
<!ELEMENT phi (rule+)>
<!ELEMENT rule (condition, then, else?)>
<!ATTLIST rule applyto (summary|file) "summary">
<!ELEMENT condition (#PCDATA)>
<!ATTLIST condition applyto (all|url|title|description|date|rank) "all">
<!ELEMENT then (rank|throw|rule)>
<!ELEMENT else (rank|throw|rule)>
<!ELEMENT rank (#PCDATA)>
<!ATTLIST rank value CDATA #REQUIRED>
<!ELEMENT throw (EMPTY)>
```

Listing 4.4: The DTD used to build the input files

The document type definition (DTD) specifies the structure of an XML file. It defines the names of the tags, the order in which they must or may appear, which tags are optional and which tags take a parameter [18]. Listing 4.4 shows the DTD for our case here.

Chapter 5

The program

5.1 Data and Control flow

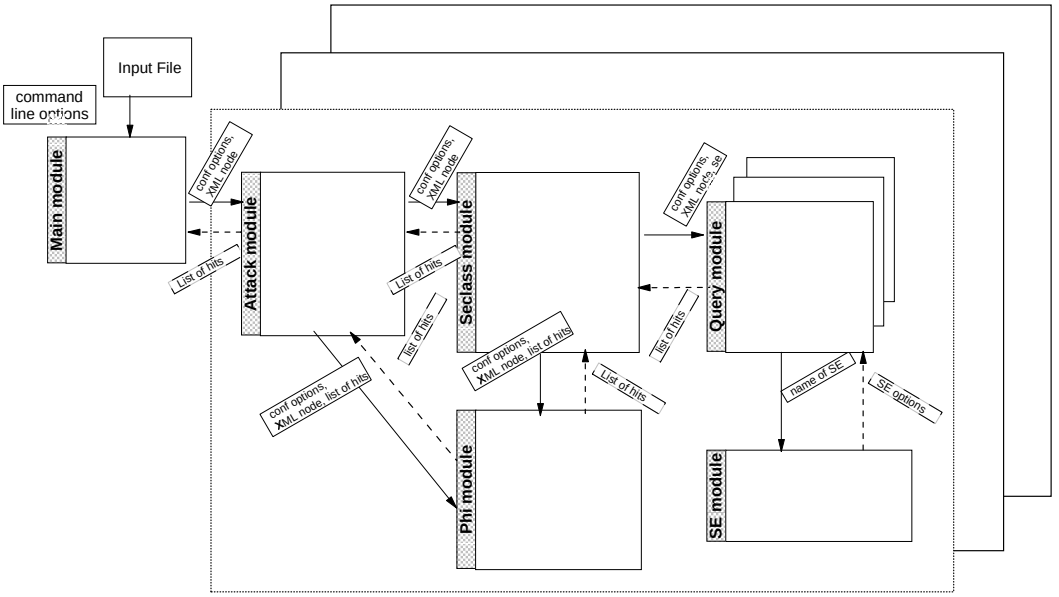


Figure 5.1: Data and Control Flow between the different modules

Figure 5.1 represents schematically the program. It is composed of six modules, and each module is responsible for one part of the input file. When a module is called, it gets as parameters the part of the Document Object Model (DOM) (cf. section 3.3.2) it is responsible for and the initial command-line options. The phi module also gets the list of hits as input, because it applies the filter over them. The module called SE, which stands for search engine, does not take a DOM parameter. It contains no active code but just some variables describing the specialities of each search engine. Besides the SE module, all modules return a list of hits to the module that called them. If a query has no results or does not work, the list will be empty.

In order to boost the performance, the program is multitasked. For each attack, a new process is forked by the main module. Another place of parallelisation is the seclass module. It forks a new process for each search engine to contact. So the treatment of one query only lasts as long as it takes to the slowest search engine. When all forked processes have finished, they send their

data to their caller process and die. Then the father process, who was waiting takes over control again. When the search engine does not respond during a certain time, the process aborts and sends an empty list to its father process. As speed was our main motivation for conceiving the tool in a parallelised version, we do not bother about the traffic bursts on the network that are a logical consequence when accessing a lot of search engines simultaneously.

In the following sections, we will look into more detail at what exactly each module is doing.

5.2 Main module

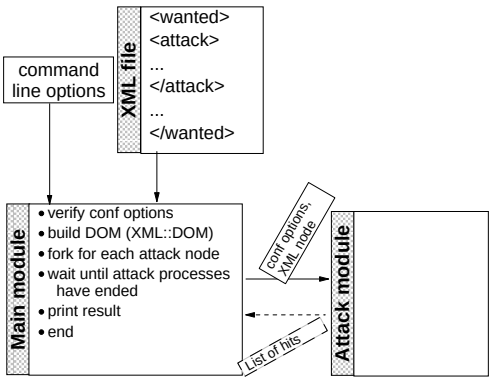


Figure 5.2: Internal functioning of the Main module

The user calls the program from the command line and gives it the name of at least one input file and several command line parameters, that allow to toggle some options of the program. Then it is the main module that begins execution. First it verifies the input parameters, and then it reads the input files (fig. 5.2). They are parsed and, when they are well-formed, the program can build the DOM. If there are more than one input file, their DOMs are appended. So there is no difference whether there is only one attack section per input file or whether all attack sections are in one input file. The resulting DOM is the same. At this stage, the module does not try to analyse the content of the input files to try an eventual optimisation (put some queries or filters in evidence), because it is unlike that there are redundancies in our input files.

When there is more than one child-node under the top-node (<wanted>), the main-module forks child processes. Each child process gets as parameter the part of the DOM that it will have to deal with and the user's command line options.

As soon as a child process finishes, the main module retrieves its results and prepares an html file for output. When all child processes have finished a title page is generated and the main process stops.

In the whole program, parallelisation is realised by calling the system routine fork. The inter process communication is assured by anonymous pipes. In chapter 6 we will discuss that a little more in detail.

5.3 Attack module

The attack module gets as parameter a sub-tree representing the attack part. It is in DOM format. The module assumes that the input file is valid and reads in the first sub node, the header. Then it checks whether there is a reference section or not. When it comes to the body part, it calls the

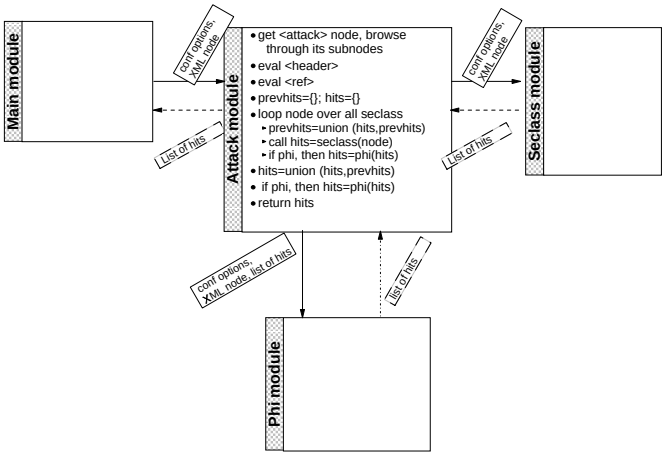


Figure 5.3: Internal functioning of the Attack module

seclass module for each seclass found (fig. 5.3). The parameters passed to the called module are always the same: the seclass-node and the command-line options.

From the seclass module it gets a list of hits. If there is a phi section after an seclass section, then the phi module is called, with that list of hits as parameter. Otherwise, the list is merged with the previous list of hits.

Once all nodes have been terminated, the whole list of hits and the header and reference information are returned to the main module.

5.4 Seclass module

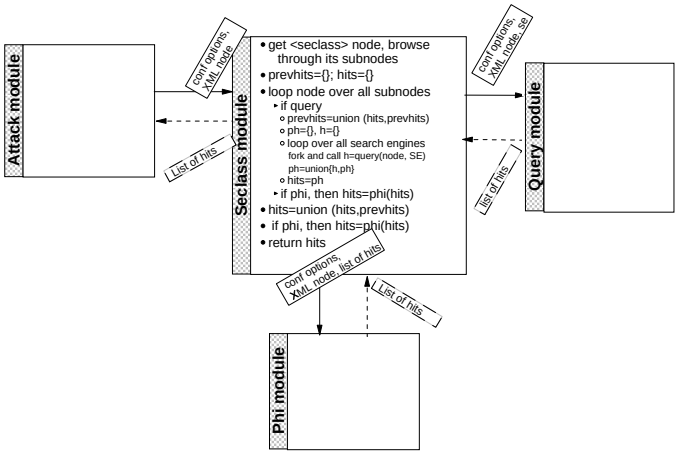


Figure 5.4: Internal functioning of the Seclass module

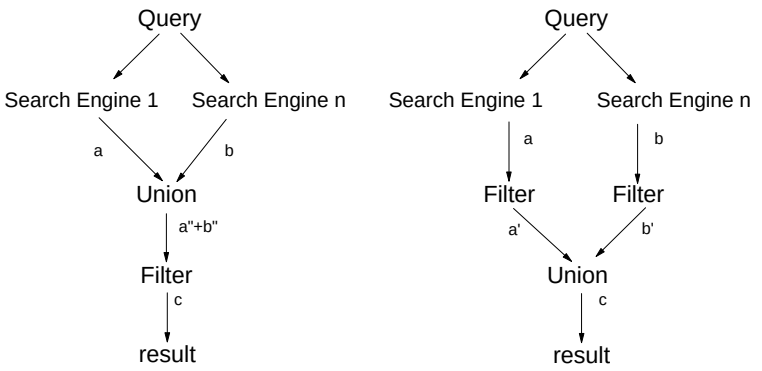
The seclass module loops through the child-nodes of the seclass node (fig. 5.4). When it detects a query node, it builds an array with all the search engines that have a class higher or equal to the parameter of the class. That allows to use special features in the query, that are not supported by all search engines. Unfortunately, as seen in table 2.1, that classification was not so easy to do, and some search engines are put in a class, even though they do not support all the options of their class. In such cases, the program makes its best effort: whenever there is an unknown

keyword for the search engine in the query, it is ignored, but the process is not aborted. That is in fact a general characteristic of the program: it is quite fault tolerant and always tries to find a result even if parts of the query must ignored.

When that array has more than one entry, a new process is forked for each entry. Each sub-process calls the query module with the name of the search engine to use and with the query node as parameter. Then the main process waits until all sub-processes have returned some results and stopped. All the results are merged together into a large list. We use an associative array to detect duplicates. When we have a hit with two times the same URL, there is a conflict. To solve it, a new hit is created with the same URL. For each field of the new hit, the module fills it up with the value of the largest entry in the corresponding field of the two hits in conflict. So we can guarantee, that in our results, all URLs are unique and hits are most complete.

Once the query is finished, the seclass continues to loop through the seclass node. When it encounters a phi bloc, it calls the phi module with the latest retrieved hits as argument and it gets back the filtered list. That list is then merged with the previously found hits. When it has finished with the seclass node, the list of results is returned to the attack module.

At a certain moment, we thought about distributing the filter behind a query and apply it directly after the results are got back from the search engine. A little calculation, with simple assumptions, shows that that would be less efficient.



A) union before filtering B) filtering before union

Figure 5.5: Schema explaining the two ways to do the filtering

Assume that applying a filter has a cost of λ. Be f the number of hits to filter. Assume that applying a union operation has a cost of ρ. Be u the number of hits that are unified.

So we want to evaluate: $\lambda \cdot f + \rho \cdot u$ for the two cases where:

1. $f = a'' + b''$ and $u = a + b$.
2. $f = a + b$ and $u = a' + b'$.

Now we can write $\lambda \cdot (a'' + b'') + \rho \cdot (a + b) \leqslant \lambda \cdot a + \lambda \cdot b + \rho \cdot (a' + b')$

Let us divide by ρ and $\lambda' = \lambda/\rho$.

$$\Leftrightarrow \lambda'(a'' + b'') + (a + b) \leqslant \lambda'(a + b) + (a' + b') \quad \text{and from fig. 5.5 :} \quad a' + b' \leqslant a + b$$

$$\Leftrightarrow \lambda'(a'' + b'') + (a' + b') \leqslant \lambda'(a + b) + (a' + b')$$

$$\Leftrightarrow (a'' + b'') \leqslant (a + b)$$

And that is correct. So it is more efficient to filter only one time.

5.5 Query module

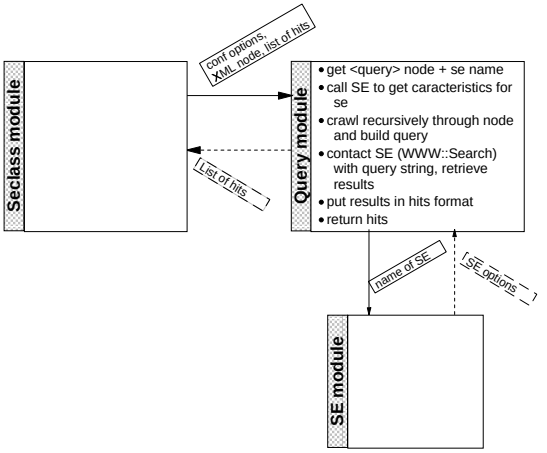


Figure 5.6: Internal functioning of the Query module

The query module is called with the query node and the name of the search engine to use as parameter. First it calls the SE module to find out the specifications of the search engine (fig. 5.6). Then it begins to analyse the query node in a recursive way. When there is an unknown tag, it is ignored. When the search engine in use does not accept boolean phrases, the query is transformed in a form with + and − before the keywords, in order to keep as much as possible of the meaning of the boolean expression.

When the query node is evaluated, the query module uses the WWW::Search module from CPAN to contact the search engine and retrieve the hits. Currently Cercator supports eight search engines: AltaVista in normal and in advanced mode, EuroSeek, Excite, GoTo, Google, HotBot, Infoseek, Lycos in normal and in advanced mode and NorthernLight. As Infoseek in advanced mode is equivalent to AllTheWeb, AllTheWeb is not supported as itself.

When the results come back from the search engines, they are put in the right format. They also get a new ranking. For each search engine, the ten first results get a ranking of 200. The ten following get a rank of 195 and so on. No hit gets a negative ranking however. This feature allows to give a higher value to those hits, that the search engine returned first.

5.6 Phi module

5.6.1 Filtering summaries

The phi module gets as parameters the phi node and the list of the hits to filter. First the condition field is treated. The module loops through the whole list of hits and for each hit, it sets the variable $x before it evaluates the condition (fig. 5.7). $x can be either the url, the title, the rank, the description or the concatenation of all these fields.

The condition is a Perl expression, that is evaluated and returns either true or false. Depending on the result of the condition, the url is put either in the then-list or in the else-list.

When all hits have been evaluated, the <then> bloc is evaluated. It applies on the then-list. When the action is <rank>, the ranking of all hits from the then-list is adjusted. When the action is <throw>, the then-list is emptied. When the action is <rule>, then filter is called again, with that rule and the then-list as parameters. The returned list is the new then-list.

When there is an <else> bloc, it is proceeded in the same way as the <then> bloc but over the else-list. At the end, the then-list and the else-list are put together and returned to the calling module.

5.6.2 Filtering files

This time the filter applies to the file. That means that Cercator first has to download the file. As in our project, we do not want to allow the access on the remote site, downloading a file is in principle forbidden. But sometimes it makes some sense for example to check if the links are still valid or not. We do not yet launch an attack when only downloading an html file.

As downloading a file is an expensive operation, we do not do it each time there is a filter in the input file, but only at the end of the program. During the normal Phi calls, hits are marked that they are due for download. Another reason for delaying the download operation is, that in the meantime some hits may have been thrown away by other filters.

So at the end of the program, the filter module is called again from the attack module with the whole list of hits as argument. Then the program loops through the list and takes out all hits that are marked for download. All these hits are put in a new list and are downloaded together. We use the LWP::Parallel module from the CPAN to download all these pages in the same time. After the download operation a loop through the list takes out all the hits where the download was successful. All other hits are put in a list for return. Now we have a list with hits and their corresponding file. Then the filters are applied as before but this time the $x variable contains the whole file.

When all hits have been evaluated, the list of the remaining hits is returned to the attack module.

5.7 SE module

The SE module is in fact not a real module, as it does not contain real code. It just contains some data necessary to translate the query from the query bloc into a search engine specific format (fig. 5.8). When a new search engine should be added, it suffices to add a few lines in this module and to add a search engine specific back end. The rest of the program will recognise the new search engine automatically. The back end can be build either by the programmer or, with a little chance, found on the CPAN in the WWW::Search module[1].

[1] http://www.isi.edu/lsam/tools/WWW_SEARCH/ and http://members.xoom.com/back ends/

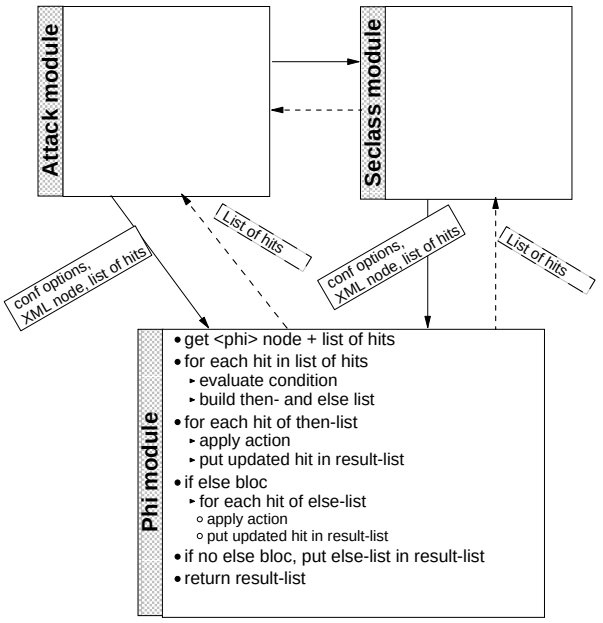


Figure 5.7: Internal functioning of the Phi module

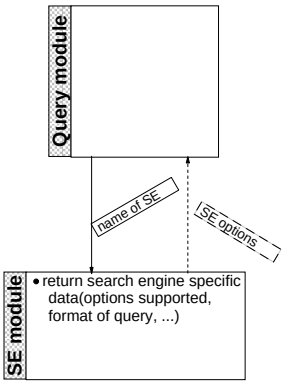


Figure 5.8: Internal functioning of the SE module

Chapter 6

Critical analysis of the program

6.1 Difference between this metacrawler and others

Cercator has a number of advantages in comparison to the classical metacrawlers. As we wrote it ourselves, we have the source code and can always modify it. Cercator is Operating System independent, because Perl can run on various operating systems. At the moment Cercator supports nine different search engines, but it is programmed in such a way, that many other search engines can easily be added without changing its code. Another major advantage is that Cercator knows what features are supported by each search engine and it is able to transpose the query specified in the input file into that special format. That characteristic was not found in any of the analysed metacrawlers in section 2.2.5.

And the last major feature is its ability to apply a filter on the results retrieved from the search engines. The condition can apply either to the description of a hit or to the file referenced by the hit. And the condition can be as complex as needed, because it is written in Perl. All these advantages make, that Cercator is more than just another metacrawler.

6.2 Known Problems

In section 2.1.1 we criticised that programmers did not verify the data before they manipulate it. Well in our case here, there is a similar problem. The conditions in the <phi> bloc are evaluated as is, even if they contain other Perl code as they should. But we think, that that does not constitute a real problem for the moment, as we suppose the tool will be used internally by knowledgeable people.

When the main module parses the XML input, it should not only verify that the document can be parsed and that it is well-formed, but it should also check its conformity with the DTD. That check does not exist in our case. But the program itself controls at many stages, whether the next tag is what it should be. If there is a problem with the input file, it is signalled, but whenever possible the program tries to ignore the problem and goes on. That is its best effort characteristic. Only in special situations, the program has to stop because of a serious error in the input file (e.g.: <Phi> preceding a <query> bloc).

In the Phi module, we use the LWP::Parallel module to retrieve in parallel the marked hits, but tests have shown, that that works rather slow at larger scales. The downloads of several hundred files lasted so long, that we think, the module does not really work fine.

Another point to note is that the search engine back ends have to be updated regularly. It is a fact, that search engines change from time to time their formats and at that moment, the

Perl code does not work any more. But as we have foreseen that problem, we designed a modular program that facilitates updates.

During stress tests of the program, we noticed, that the operating system forbids to have more than 64 sub-processes (at least in our case). In that case, the program will not work correctly any more and the parallelisation, as it has been implemented for the moment, is useless. That problem comes from the OS, but can be bypassed by running the program without the fork option or at least in less severe conditions: either with less attack files or with less search engines. It could be definitely resolved, when realising the parallelisation in a different way as with forks.

6.3 Improvement suggestions

No program is ever finished and it is never perfect and so is this here. There are a number of possibilities to improve the program. At the moment of parsing, a DTD check should be performed. That is not too difficult. In the XML::DOM module are methods for doing a DTD conform parsing. It would also be interesting to add a few more search engines, for example Raging or Yahoo or private search engines that only index pages of a company's local network.

The relatively poor speed performance of the program is not due to Perl, because even if Perl is an interpreted language, it is still very fast. Most of the time is lost when contacting the search engines and waiting for their results. The performance can still be improved by fine-tuning the parallelisation. But Perl (at least in the version 5.005_03) constitutes a problem when trying to write multithreaded programs. The language has no native support for multiple threads, and the only way of doing is to use the fork system call. Fork is a heavy operation for the operating system and it creates multiple independent processes, and that is not what we need. So we have to use the inter process communications (IPC) support offered by the OS to communicate between independent processes. In this case, that are anonymous pipelines.

For the latest Perl version 5.6.0 issued in April 2000, the Perl developers announce a drastic reworking in the threading capabilities [19]. But the state is only experimental that means, that in the next version it can be quite different. The main problem lies in the fact that it is extremely hard for an interpreted language to support multiple threads, as the interpreter will also have to control the scheduling of each thread.

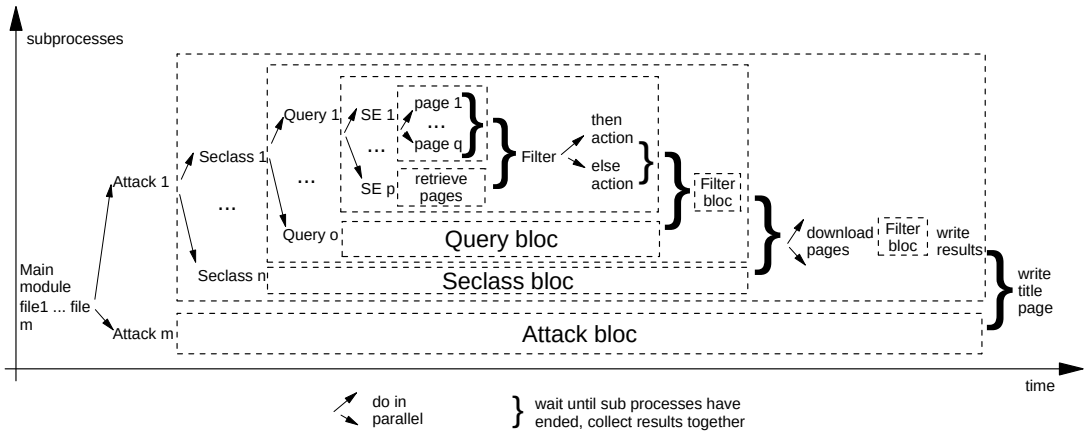


Figure 6.1: Illustration of all possibilities to parallelise

Once there is better support for threads, the program could be parallelised at various other levels. Figure 6.1 illustrates the various levels of parallelisation. The LWP::Parallel module could be adapted in order to work more efficiently. If inside one attack, there are several seclass blocs, they could be accessed simultaneously. The same is true one level below: the different queries and

their associated filters that lie inside the same seclass can also be executed simultaneously. At that moment, all queries from the whole input file would start at the same moment, and the whole process would work in a bottom up way, whereas it is more like top down at the moment. Once all queries from inside one seclass have finished, they are collected together. Once all seclasses have finished, their results are unified into one list and so on. All these parallelisations could lead to a real boost in performance, when the input files are big. But it often arrives, that there are only one or two queries in one attack, and in that case, the benefit becomes unimportant.

Another module, where tasks can be executed in parallel is the phi module. Rules always have to be executed sequentially. But filter actions can be executed simultaneously. If the then-list and the else-list are of similar sizes, then it could be interesting to execute the actions in the same time.

A module where parallelisation can be applied is the WWW::Search module. For the moment, the module sends a query to the search engine and retrieves the results. When it detects the words 'next page' it takes out that url and sends a new request to the search engine. So it can arrive, that the module has to make up to 50 requests to get 500 hits from one search engine! That is where the program in its current version looses most time, because a request normally takes a couple of seconds. It would be more interesting to be able to foresee the address of the next pages and not take it out of the results page. But that is not easy at all, because each search engine has its very own format for coding the next page. And this modification would require to rewrite a major part of the WWW::Search module. There is a chance however, that performance would become drastically increased.

The disadvantage of all that parallelisation and realising all queries simultaneously is the traffic burst generated on the network. If that becomes a serious problem, then artificial delays have to be added again.

Chapter 7

User manual

Assume that we are in a Unix environment where a standard version of Perl and the modules WWW::Search, Getopt::Long, Storable, LWP::*, XML::* are installed. These modules can be found on the CPAN[1]. Cercator runs in text mode and is started from the command line by invoking it with *perl Cercator.pl*. The only mandatory argument is the name of one XML input file. All the other options can be specified on the command line or in a configuration file. By default the configuration file is called Cercator.conf. In the configuration file, option names must be completely written out. The option name is followed by a "=" and then comes the option value. Only one option can be defined per line.

Chapter 4 explained in detail the format of the input file. In the following section we present a graphical user interface, that will greatly facilitate the writing of the files. Here we will explain the options that are recognised. Options can be specified in any order by "-- + option name". The option name can even be abbreviated as long as it keeps a unique meaning.

--class specifies the minimal class a search engine must belong to. It is used to limit the search engines to consider during a particularly run. So "class 3" takes only search engines of class 3 or 4. This parameter does not influence the queries that are executed. That means even if a query belongs to a seclass 1 bloc, it will be taken into account.

--debug specifies the amount of debug information liked. Values can range from 0 for no debug information up to 5. It is not a good idea to choose a high value for debug when using the option fork. There arise conflicts when several processes want to write their debug information in parallel.

--domain specifies the top level domain to concentrate on in this run. This option works only with search engines belonging to a class higher than 1. Otherwise it is not taken in account. Example: domain .ch

--engine specifies which search engine should be used in this run. The option can be repeated. Example: engine AltaVista engine Lycos. Recognised names are: AltaVista, AltaVista:-:AdvancedWeb, EuroSeek, Excite, GoTo, Google, HotBot, Infoseek, Lycos, LycosSimple and NorthernLight.

--file allows to specify the name of the configuration file. Command line definitions have priority over definitions from the input file. Example: file Cercator.conf

--fork allows to specify whether Cercator should run strictly sequentially (fork=0, default), whether it should fork in the seclass module for each search engine to use (fork=1) or whether it should also fork for each attack (fork=2) in the main module. Fork=2 is critical,

[1] Comprehensive Perl Archive Network at http://www.cpan.org

because the technical limit of 64 processes is reached quickly, when using a lot of input files and a lot of search engines in the same time.

--help or -? prints a little help text.

--nbresults specifies the number of results that Cercator should try to retrieve from each search engine. Some search engines have a limit (e.g. AltaVista limits at 200), but some other allow to download all hits that were found.

--output specifies the name of the output files. Default: output=results

--path specifies the path where the result files will be written. Example: path ~/temp

--permission specifies which files may be downloaded in a filter section. That option is related to the feature that allows to download the whole file and apply a filter over it.

When permission=0, no file can be downloaded whereas permission=3 gives full rights. permission=1 and permission=2 are two intermediate solutions for allowing just static html files (extensions .htm, .html) or static and dynamic html files (extensions .htm, .html, .asp, .dhtml) to be downloaded.

But permission can also be a Perl regular expression, that is checked against the URL of the file to download and defines whether or not the file may be accessed. Example: permission= m:.ibm.com/: $\longrightarrow$ everything from ibm.com can be downloaded.

7.1 Graphical User Interface

The structure of the input file is not so easy and it becomes hard to write an input file from scratch correctly with all tags opening and closing at the right places. That is why Charles-Emmanuel Daviet, another student who passed a short internship at IBM, wrote a Graphical User Interface, that makes the creation of input files an easy task [20].

For each recognised tag there is a button that inserts the opening and closing part of it in the editor. When a tag takes a parameter, it can be specified in a pull-down menu or in a dialogue box. For a better readability, tags are automatically indented and coloured. The GUI allows to build an input file only by clicking with the mouse and completing with the right keywords. Input files can be build from top to bottom i.e. from <wanted> to <query> or from bottom-up i.e. beginning with the queries and building the rest of the file around. Normally the keywords are just ordinary text. But there are two special cases to be explained in further detail. When working with dates, a date specified between the <fromdate> or <todate> tags has to be given in dd/mm/yyyy format. And secondly, certain characters have to be transcribed in XML. For example '<' and '>' become '<' and '>' respectively and '&' becomes '&'.

The GUI is written in Tcl/Tk and it is therefore platform independent as long as there is a Tcl/Tk interpreter available.

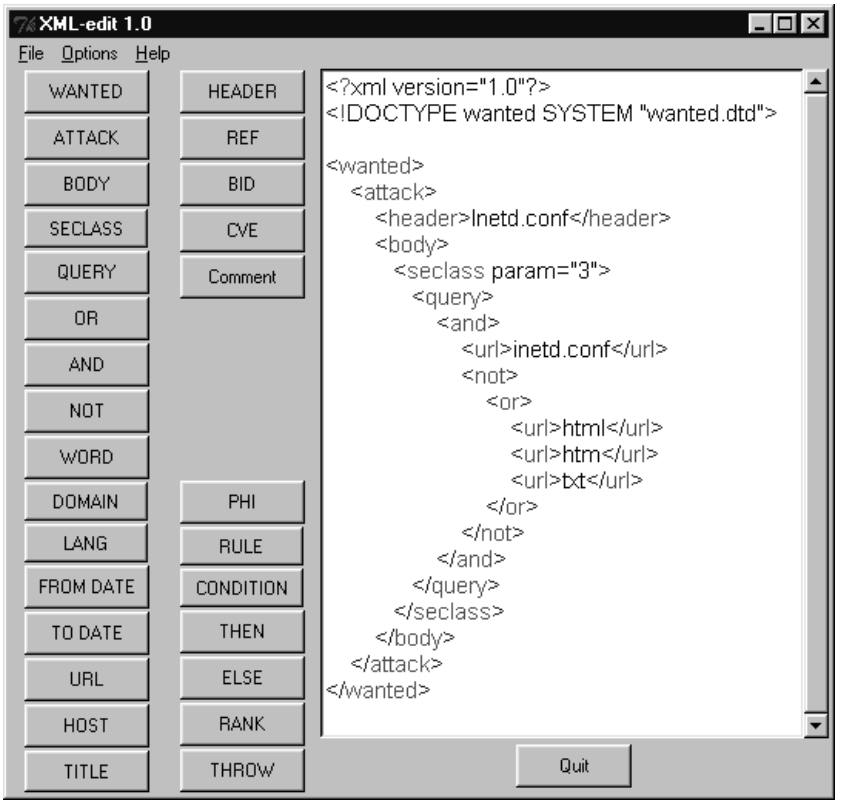


Figure 7.1: Screen shot of the GUI for creating the XML input files

Chapter 8

Input File Examples

After the detailed presentation of Cercator in the previous chapters, we will now use the program on a concrete subject: we will try to find vulnerable sites. In this chapter, we will present a non exhaustive list of vulnerabilities that Cercator can find. These examples have been chosen in order to present a lot of different ways to look for vulnerable sites and to illustrate the richness Cercator offers. The following vulnerabilities examples and the corresponding Input Files in XML were found and written in common by Charles-Emmanuel Daviet [20] and the author.

8.1 Security files and configuration files

Configuration files trigger the working of some programs and contain sensible data. In general those files are not readable from the web, but due to a misconfiguration, it arrives, that they become visible and search engines can index them. Sometimes only the knowledge of the presence of the file is a precious information for an intruder. It may simply tell him, that a given program is installed on the host. In other cases it would be necessary to access the file and read its content to gain useful information. The fact, that such files are readable, shows that the administrator of that site is not very attentive and that there may be other security lacks. The following paragraphs will present a few examples of such files that we have considered in this work.

8.1.1 Password file

The /etc/passwd file is a classical example of a security file. It contains all user logins and sometimes even the passwords. In general passwords are shadowed and they are stored in a different file. When a hacker can access the password file, he will try to decrypt the passwords with a cracker program. When the passwords are shadowed, he cannot crack them any more, but, as he has the user logins, he can still try to guess a password and so he could try to use that information to break into the system. Another possibility can be a brute force attack or in some special situations unshadowing the passwords.

In any case one can say, password files should newer be visible from the web, and when they are readable nevertheless, there is a misconfiguration problem. Sometimes there are typical users in the password file that give information about the software installed. For example certain IDS systems run under their own user identity. Cercator could also find this without having to access the file.

8.1.2 inetd.conf

Inetd is a daemon or background process that starts up at the beginning of the boot sequence. Inetd listens on many ports, and when a connection to a port is requested, it starts up the process associated with that port. Examples of services run from inetd are ftp, telnet, finger, pop, imap. All the configuration of the daemon is done via the configuration file inetd.conf. It is also considered a system security file. When a hacker can read it, he gains a lot of information about the open ports. He will see if the servers are run via a wrapper program and if there is a logging tool. But the inetd.conf does not replace the results a port-scan could give, because there can still be some services running without inetd.

8.1.3 sendmail.cf

Sendmail.cf is the configuration file of the sendmail program. Sendmail is the server side of SMTP (Simple Mail Transfer Protocol), and seems to be the predominant software used in UNIX e-mail systems. Using sendmail, mail can be delivered to individual users, distributed to mailing lists, sent automatically to another machine, appended to files, and provided as standard input to programs.

Earlier versions of sendmail are known for having been plagued with many security problems. One of the main reasons for sendmail's problems is its all-in-one design. The program is extremely complicated, runs as super-user, freely accepts connections from any computer on the Internet, and has a rich command language. So when a sendmail.cf file is present, we can suppose that the sendmail program is used.

8.1.4 .rhosts

A .rhosts file with the names of some hosts or even whole sub networks in someone's home directory allows him to login to any machine enumerated without specifying his password. If ~/.rhosts contains: *+ name*, any host with a user *name* can rlogin your account without supplying the password. If ~/.rhosts contains: *host +*, any user initiating a connection from the machine *host* can "rlogin host -l name" without supplying a password. Hence, it is quite risky to use the .rhosts feature in Unix.

8.1.5 Httpd.conf, access.conf and htaccess files

Httpd.conf is the main configuration file of the web server. As the file is used at least by Apache and NCSA HTTPd, it gives only limited information about the server in use. It defines how httpd behaves, how it is started, on which port it listens, under which user ID it should run, in which directories lie the CGI scripts and so on.

Access.conf also belongs to the web server and defines the access control to be enforced on the server: it contains critical information about who has which access rights to which directories of the server. The content of the file gives information about known user names and – up to a certain level – about the directory structure on the remote host. For example, branches considered unsafe (such as users' home directories) can be made more secure by disabling certain server functions in those directories.

The htaccess file is needed when working with pages, that are password protected. It controls the access per user of the directory it lies in.

8.1.6 Log files

The files var/log/messages, var/log/access_log, /var/log/error_log and var/log/errors are examples of common system files, that contain logging information about the system. The web server uses two log files for himself. In httpd.log it writes all successful accesses to the server whereas in error_log it writes all failures. As these files contain critical information about the installed software and even about IDS systems. They should not be available to public access. In any case, it makes no sense to publish them.

8.1.7 Description of the searching strategy for configuration files

When looking for security or configuration files, there are two ways to do it: either one uses a powerful search engine that supports the url tag and one looks directly for files having the right name. Otherwise, one has to study the internal structure of the file and pick out several words that are very typical. Then one looks for these keywords and one applies a filter to check whether the name of the file is really what one is interested in.

When the input file is well built, the search returns links to a file having the name we are looking for. But when that file lies in a directory called examples or doc or something similar, it is likely, that the hit is a wrong hit for our context, because it is not the real security file that is referenced. It is only a file with that name, sometimes even with the right structure, but not in the right directory and therefore not used by any program.

The example in listing 8.1 is the description of two queries that look for access.conf files. For search engines of class 1, only the two keywords 'Directory' and 'access.conf' are searched. The following filter is necessary to keep only those hits, where access.conf is at the end of the url. For search engines of class 3, the query is different. The keyword url allows to precise the query and to search only for files that contain access.conf in the url, but the files should not be htm, html or txt files. It is important to note here, that several search engines do not support negated URL clauses. In that case, Cercator simply ignores these parts of the query. That can have as consequence that the results seem rather wrong, because they do not verify one clause in the query.

In listing 8.2 we present our input file that looks for password files. It checks for the word root and the url /etc/passwd. The filter that follows increases the rank of those hits, that do not have their password shadowed.

8.2 AltaVista Search Engine bug

8.2.1 Description of the problem

AltaVista offer a software to run a search engine with the same performance and efficiency as the well known AltaVista search engine to anybody for its private intranet. A bug in the main search function of several versions of the software allows everyone to traverse one step back in the directory structure and so to gain access to a directory containing administrative information of the machine hosting the search engine, including the password for the remote administration utility.

8.2.2 Description of the searching strategy

It is rather difficult to use the search engines to search for sites that have an internal search engine powered by AltaVista. There is no common point that would allow to give keywords. So this bug allows a lot of different searching strategies. Listing 8.3 gives an example how we can start. It looks for some keywords that describe AltaVista. The query is followed by a complex filter that

```
<?xml version="1.0"?>
<!DOCTYPE wanted SYSTEM "wanted.dtd">

<wanted>
    <attack>
        <header>Access.conf file</header>
        <body>
            <seclass param="1">
                <query>
                    <and>
                        <word>Directory</word>
                        <word>access.conf</word>
                        <word>server configuration file</word>
                    </and>
                </query>
                <phi>
                    <rule applyto="summary">
                        <condition applyto="url">
                            $x!~/access\.conf$/
                        </condition>
                        <then>
                            <throw/>
                        </then>
                    </rule>
                </phi>
            </seclass>

            <seclass param="3">
                <query>
                    <and>
                        <url>access.conf</url>
                        <not>
                            <or>
                                <url>html</url>
                                <url>htm</url>
                                <url>txt</url>
                            </or>
                        </not>
                    </and>
                </query>
            </seclass>
        </body>
        <phi>
            <rule applyto="summary">
                <condition applyto="url">
                    $x=~/examples?/i || $x=~/howto/i || $x=~/doc/i
                </condition>
                <then>
                    <throw/>
                </then>
            </rule>
            <rule applyto="summary">
                <condition applyto="all">
                    $x=~/Global access configuration/i || $x=~/server configuration file/
                </condition>
                <then>
                    <rank value="+200"> </rank>
                </then>
            </rule>
        </phi>
    </attack>
</wanted>
```

Listing 8.1: Input File for access.conf vulnerability

```
<?xml version="1.0"?>
<!DOCTYPE wanted SYSTEM "wanted.dtd">

<wanted>
    <attack>
        <header>/etc/passwd file</header>
        <body>
            <seclass param="3">
                <query>
                    <and>
                        <word>root</word>
                        <url>/etc/passwd</url>

                        <!-- Here we try to eliminate all the html or txt files, in order to keep only
                the real passwd files -->
                        <not>
                            <or>
                                <url>html</url>
                                <url>htm</url>
                                <url>txt</url>
                            </or>
                        </not>
                    </and>
                </query>

                <phi>
                    <rule applyto="summary">
                        <condition applyto="url">
                            $x!~/(etc\/passwd)$/
                        </condition>
                        <then>
                            <throw/>
                        </then>
                    </rule>
                </phi>

            </seclass>
        </body>
        <phi>
            <!-- Filter to increase the importance of those hits, where password is not shadowed -->
            <rule applyto="summary">
                <condition applyto="description">
                    $x!~m@root:.?:@
                </condition>
                <then>
                    <rank value="500"/>
                </then>
            </rule>
        </phi>
    </attack>
</wanted>
```

Listing 8.2: Input File for /etc/passwd vulnerability

changes the ranking of the results depending on the presence or absence of certain keywords. At the end, we will get a results file, where the best matching results are given first.

8.3 Web servers

8.3.1 Description of the problem

Many widely used web servers have bugs that may become very dangerous for the hosts where they are installed on. Certain versions of the Apache Web server have a buffer overflow error.

The Internet Information Server (IIS) from Microsoft is also a widely used web server. Many vulnerabilities have been detected in IIS. The most recently[1] detected one concerns the default (error) pages of IIS. Another error is the vulnerability in IIS 4.0 to a buffer overflow that allows either to crash the server or to run arbitrary commands on it.

And as a final example let us cite the Microsoft Personal Web Server that comes along with Microsoft Windows 98. This program has a bug that allows to access the root directory of the host.

These are only a few examples that illustrate the problems there are, when not using the most recent version where all patches have been applied.

8.3.2 Description of the searching strategy

There is no real way to detect with certainty the type of web server a site is using without contacting it. But with a few assumptions, it becomes possible to get some results nevertheless. After a standard installation of the Web server, its html documentation gets installed too and is visible from the web. So search engines can index it. The hypothesis in this strategy is the following: we assume, that when the standard documentation for a particular web server is installed, then that server in the corresponding version is running as well.

This strategy, consisting of searching standard documentation of a web server, does not find all sites running the respective server, but it finds a lot of them. The strategy will not give an exhaustive result. But on the other side the presence of the Apache documentation version 1.2 for example, does not tell whether it is still running or whether it has been patched or even been updated to a more recent version. So the results in this case give only indications on potentially vulnerable sites. If the links are broken, the conclusion becomes more uncertain. We can only say, that some time ago, the link existed, and that at that time the documentation was available. It can be quite probable, that the same server is still running, and the administrator has just changed its configuration files in order to make the server manuals unavailable from the web.

In any case, this strategy will give us a lot of results, but the value of the results is not very high for us. We want to find vulnerable sites but this strategy will deliver to us only sites that offer the standard documentation for a web server.

Listing 8.4 shows how to find the standard documentation of MS-IIS. There are simply several queries that look for certain typical sentences that can only be found on these pages. At the end a filter will keep only results with a specific keyword.

[1] by the time this writing takes place (June 2000)

```
<?xml version="1.0"?>
<!DOCTYPE wanted SYSTEM "wanted.dtd">

<wanted>
    <attack>
        <header>AltaVista Search Engine 2.0b, 2.3A, V2.3A </header>
        <ref>
            <!--reference for this attack is bugtrack id 896 -->
            <BID>896</BID>
        </ref>
        <body>
            <seclass param="1">
                <query>
                    <and>
                        <word>search</word>
                        <word>AltaVista Search Intranet</word>
                        <not>
                            <word>Installation and Configuration</word>
                        </not>
                        <or>
                            <word>Query</word>
                            <word>Advanced</word>
                        </or>
                    </and>
                </query>
            </seclass>
            <phi>
                <rule applyto="summary">
                    <condition applyto="title">
                        $x=~/AltaVista Search Intranet V\d\.\d/i ||
                        $x=~/AltaVista Search Intranet/ ||
                        $x=~/Search: Simple Query/
                    </condition>
                    <then>
                        <rank value="300" />
                    </then>
                    <else>
                        <rule applyto="summary">
                            <condition applyto="url">
                                $x=~m!http://.*/$! || $x=~m!http://w0,3\.?search\..*/! ||
                                $x=~m!http://.*/av/content/! || $x=~m!http://.*/search\.html?$! ||
                                $x=~m!http://.*?:\d\d+/! || $x=~m!http://.*/version\.html$!
                            </condition>
                            <then>
                                <rank value="200" />
                            </then>
                            <else>
                                <throw/>
                            </else>
                        </rule>
                    </else>
                </rule>
                <rule applyto="summary">
                    <condition applyto="all">
                        $x=~m!v2\.3!i ||  $x=~m!Intranet 97! || $x=~m!eXtension 97!
                    </condition>
                    <then>
                        <rank value="200" />
                    </then>
                </rule>
                <rule applyto="summary">
                    <condition applyto="url">
                        $x=~m!http://.*/\w*\d+\.html?$!
                    </condition>
                    <then>
                        <throw/>
                    </then>
                </rule>
            </phi>
        </body>
    </attack>
</wanted>
```

Listing 8.3: Input File to search for AltaVista search engine

```
<?xml version="1.0"?>
<!DOCTYPE wanted SYSTEM "wanted.dtd">

<wanted>
    <attack>
        <!-- Microsoft Internet Information Server default page -->
        <header>MS-IIS</header>
        <body>
            <seclass param="1">
                <query>
                    <and>
                        <word>You can manage the IIS services from a Web browser</word>
                        <word>To learn more about how Internet Information Server</word>
                    </and>
                </query>
            </seclass>
            <seclass param="2">
                <query>
                    <and>
                        <word>You can manage the IIS services from a Web browser</word>
                        <word>To learn more about how Internet Information Server</word>
                        <title>Microsoft Internet Information Server</title>
                    </and>
                </query>
                <query>
                    <and>
                        <word>Microsoft Internet Information Server</word>
                        <title>Internet Service Manager</title>
                    </and>
                </query>
            </seclass>
            <seclass param="3">
                <query>
                    <and>
                        <word>Microsoft Internet Information Server</word>
                        <url>default.htm</url>
                    </and>
                </query>
            </seclass>
        </body>
        <phi>
            <rule applyto="summary">
                <condition applyto="all">
                    $x=~m@samples?@i || $x=~m@default@i ||
                    $x=~m@Microsoft Internet Information Server@i ||
                    $x=~m@iisadmin@i
                </condition>
                <then>
                    <rank value="250"/>
                </then>
                <else>
                    <throw/>
                </else>
            </rule>
        </phi>
    </attack>
</wanted>
```

Listing 8.4: Input File to search for MS-IIS servers

8.4 Active server pages

8.4.1 Description of the problem

Active server pages (ASP) with runtime errors expose a security hole that publishes the full source code name to the caller [21]. If these scripts are published on the Internet before they are debugged by the programmer, the major search engines index them. These indexed ASP pages can be located with a simple search. The search results publish the full path and file name for the ASP scripts. This URL can be viewed in a browser and may reveal full source code with details of business logic, database location and structure.

The solution to the problem, that is not a software bug, is to prevent the search engines from indexing pages that have ASP runtime errors. In section 2.2.1 we presented the solution with the robots.txt file. Furthermore programmers should fully debug their ASP scripts before publishing them on the web.

8.4.2 Description of the searching strategy

It suffices to build a query that searches for "+Microsoft VBScript runtime error +.inc". Listing 8.5 shows the final file.

8.5 Cart32 shopping cart software

8.5.1 Description of the problem

Cerberus Information Security, Ltd[2] has discovered a serious back door in the Cart32 shopping software, which runs on Windows 95 and Windows NT machines. A secret password allows someone connecting to a web site running "Cart32" shopping cart software to gain access to the server. The server can reveal such data as credit card numbers, order information, and shipping addresses. Hundreds of small to medium web sites use Cart32. To gain access to customer files, an attacker could use the password to alter the shopping cart to leak information when users connect to the site. Cerberus said it also discovered a way to change Cart32's administrative password without knowing what the original one was.

8.5.2 Description of the searching strategy

The strategy is quite simple. First one has to check one or more sites where one knows, they use the software in question. Then one looks for typical words or elements that are present at each site. After that, it becomes easy to write an input file. In the present case, the typical phrase is "Shopping cart powered by Cart32". Listing 8.6 shows the corresponding input file.

8.6 Common Gateway Interface related problems

8.6.1 Count.cgi

Count.cgi is a popular cgi program that displays the number of raw hits on web pages as an inline image and people use it to keep track of how many hits their web pages have received. One version was plagued with two buffer overflow vulnerabilities, and any user could also read any GIF image on the server.

[2] http://www.cerberus-infosec.co.uk

```
<?xml version="1.0"?>
<!DOCTYPE wanted SYSTEM "wanted.dtd">

<wanted>
    <attack>
        <header>Active server pages with runtime errors</header>
        <body>
            <seclass param="1">
                <query>
                    <and>
                        <word>Microsoft VBScript runtime error</word>
                        <word>.inc,</word>
                    </and>
                </query>
            </seclass>

            <!-- Here we try to keep only "real" errors which give a full path
            and file name of ASP scripts-->
            <phi>
                <rule applyto="summary">
                    <condition applyto="description">
                        $x!~/Microsoft VBScript runtime error/ || $x!~/\.inc/
                    </condition>
                    <then>
                        <throw/>
                    </then>
                </rule>
                <rule applyto="summary">
                    <condition applyto="url">
                        $x=~m:/default\.asp$:
                    </condition>
                    <then>
                        <rank value="150"> </rank>
                    </then>
                </rule>
                <rule applyto="summary">
                    <condition applyto="description">
                        $x=~m:File|Path not found:
                    </condition>
                    <then>
                        <rank value="250" />
                    </then>
                    <else>
                        <rank value="-150" />
                    </else>
                </rule>
            </phi>
        </body>
    </attack>
</wanted>
```

Listing 8.5: Input File to search for ASP pages

```
<?xml version="1.0"?>
<!DOCTYPE wanted SYSTEM "wanted.dtd">

<wanted>
    <attack>
        <header>Cart32.exe</header>
        <body>
            <seclass param="1">
                <query>
                    <or>
                        <word>Shopping cart powered by Cart32</word>
                        <word>Shopping Cart</word>
                        <word>McMurtrey/Whitaker</word>
                    </or>
                </query>
            </seclass>

            <seclass param="3">
                <query>
                    <url>cart32.exe</url>
                </query>
            </seclass>
        </body>

        <phi>
            <rule applyto="summary">
                <condition applyto="url">
                    $x!~m:/cart32\.exe/:
                </condition>
                <then>
                    <throw></throw>
                </then>
            </rule>
            <rule applyto="summary">
                <condition applyto="description">
                    $x=~/Cart32 v\d\.\d/
                </condition>
                <then>
                    <rank value="500"></rank>
                </then>
            </rule>
            <rule>
                <condition applyto="all">
                    $x=~/Build \d+/i
                </condition>
                <then>
                    <rank value="500" />
                </then>
            </rule>
            <rule>
                <condition applyto="all">
                    $x=~/ItemList/i
                </condition>
                <then>
                    <rank value="300" />
                </then>
                <else>
                    <rank value="-100" />
                </else>
            </rule>
        </phi>
    </attack>
</wanted>
```

Listing 8.6: Input file to search sites using cart32

8.6.2 Glimpse

Some vulnerabilities exist in the GlimpseHTTP and WebGlimpse packages. Both of these packages provide a web interface which allows you to use Glimpse, an indexing and query system, to provide a search facility for your web site. The cgi-bin programs in these packages perform insufficient argument checking. Due to this, intruders may be able to execute arbitrary commands with the privileges of the httpd process.

8.6.3 Description of the searching strategy

For these two attacks there is no more a general rule how to build the queries. But the idea is to find typical elements in the URLs or on the pages of the sites using the particular program. For count.cgi it is relatively easy. The keyword is simply count.cgi in the url or count.cgi as keyword itself. As not every version has the bug, it would be preferable to limit the search only to the vulnerable versions of the program. To achieve that, we use the date of modification of the page. With the date clause we will find only the vulnerable version 2.3 that came out on Oct 17, 1997 and lasted until Sep 14, 1998, release date of version 2.4 [22].

For Glimpse the problem is more difficult. But we have noticed, that a lot of the sites using WebGlimpse use the same input interface, and so we can look precisely for that input mask. Listing 8.7 shows the resulting input file.

```
<?xml version="1.0"?>
<!DOCTYPE wanted SYSTEM "wanted.dtd">

<wanted>
    <attack>
        <header>GLIMPSE</header>
        <body>
            <seclass param="1">

                <query>
                    <and>
                        <word>WebGlimpse Search</word>
                        <word>Glimpse and WebGlimpse</word>
                        <word>University of Arizona</word>
                    </and>
                </query>

                <phi>
                    <rule applyto="summary">
                        <condition applyto="all">
                            $x=~/(updated|revised|modified)/
                        </condition>
                        <then>
                            <rank value="80"></rank>
                        </then>
                        <else>
                            <rank value="-50"> </rank>
                        </else>
                    </rule>
                    <rule applyto="summary">
                        <condition applyto="all">
                            $x=~/wgindex\.html/ || $x=~/Full search on/  || $x=~/ghindex\.html/
                        </condition>
                        <then>
                            <rank value="150"> </rank>
                        </then>
                        <else>
                            <rank value="-30"> </rank>
                        </else>
                    </rule>
                </phi>
            </seclass>
        </body>
    </attack>
</wanted>
```

Listing 8.7: Input file to search sites using WebGlimpse

Chapter 9

Experiences and results

Once a certain set of the input files was completed and returned in majority valid results, we did a test series over a week. We were interested to see if the results obtained by Cercator changed with time. We started Cercator with the input files as parameter. The nbresults option was set to 300 and fork was 1. As there were no other options Cercator automatically used all search engines it supports at the moment.

Table 9.1 shows how many results we got back for each input file. For certain attack files, the number of results stays nearly constant (access.conf, web server log dirs, asp, httpd.conf, inetd.conf, var/log/messages, MS-PWS, .rhosts, sendmail.cf and srm.conf). For other files however there is a big variation. It is not easy to explain that difference, but its origin must be due to the search engines and not to Cercator since Cercator was not changed during that week and it ran each day on the same machine under the same conditions. In order to analyse in more detail, which search engines have a large variation and which search engines do not return any interesting results, we analyse the results files in more detail.

The results of these analyses are presented in tables 9.2–9.11. They show, that the same search engine does not always return the same number of results on two following days. Normally the difference is rather small, but not always. A small difference can be explained by the fact, that the content of the search engines' database is constantly changing. However, the update is generally done relatively slowly. It takes them about three months for renewing a database of several hundreds of million entries.

The other explanation is, that the big search engines run on more than one machine in order to improve the performance and the safety of their service. Our tests lead us to believe that these machines do not share exactly the same database. AltaVista for example returns the same results on June 14 as on June 16, but on June 15 the results are different for the attack access.conf (table 9.2). The search engines that run on more than one machine for load balancing reasons are AltaVista and Lycos, because they have several IP addresses.

Another comment concerns the efficiency of certain search engines. At the moment Cercator supports search engines that return almost no valid results. These are Excite with only one result for the AltaVista attack, GoTo and HotBot with no results at all. That means, that these search engines do not need to be used in future because they will not return any valid results (at least for the input files, that we have at the moment). This observation means, that these search engines are of poor quality, in this context. Their bad result can be partly explained. In nearly all our input files, we use the URL tag, either in the query for search engines of class three and four or in the filter blocs. HotBot belongs to class four but it is an exception (cf. table 2.1). It does not understand the URL keyword. So these queries are skipped and that is one reason for the results from HotBot.

GoTo has the speciality, that the results returned by the search engine do not contain the real

	June 13	June 14	June 15	June 16	June 19
Access.conf	7	7	4	7	7
AltaVista	370	418	419	407	428
Apache	1055	1054	1057	1044	1024
Log dirs	585	578	582	580	574
ASP pages	325	341	336	335	325
Cart32	586	550	401	393	582
Count.cgi	358	552	552	552	552
Glimpse	955	1052	1068	893	998
htaccess	96	156	137	155	156
httpd.conf	74	74	67	73	78
inetd.conf	7	10	11	13	13
var/log/messages	9	10	10	9	9
MS-IIS	2521	2567	2573	2534	2539
MS-PWS	253	251	253	253	251
etc/passwd	255	276	78	297	273
.rhosts	6	6	5	5	6
sendmail.cf	5	2	5	5	5
srm.conf	1	7	1	6	6

Table 9.1: Number of results returned by each search engine for all our test files

AltaVista	June 13	June 14	June 15	June 16	June 19
Access.conf	4	4	1	4	4
AltaVista	154	156	157	157	165
Apache	370	370	370	370	370
Log dirs	0	0	0	0	0
ASP pages	82	88	87	87	82
Cart32	260	261	261	260	261
Count.cgi	0	0	0	0	0
Glimpse	200	200	200	200	200
htaccess	81	116	116	116	116
httpd.conf	16	12	5	12	17
inetd.conf	1	4	5	7	7
var/log/messages	3	3	3	3	3
MS-IIS	634	662	656	630	632
MS-PWS	28	28	28	28	25
etc/passwd	199	199	78	199	198
.rhosts	6	6	5	5	6
sendmail.cf	5	2	5	5	5
srm.conf	0	5	0	5	5
Sum	2043	2116	1977	2088	2096

Table 9.2: Detailed analysis of the results returned by AltaVista

URL in the URL field, but they point to the GoTo site. GoTo will translate them in the real URL at the moment of accessing the file. That means, that even if GoTo returned a result, it would fail in our filters when the URL is checked. All the other search engines return results for all the attacks they can evaluate (according to their class).

EuroSeek shows a strange behaviour. It has the same number of results during three days and then it does not return any results any more (cf. table 9.4). We can only explain that fact, with a change in the internal format of EuroSeek. So our search engine specific back end cannot send off the query correctly or it cannot parse the results returned by the search engine any more. That demonstrates how important it is to check regularly that all back ends are still working correctly.

As a last remark we can say, that AltaVista is one of the best search engines in our test. It returns 44 % more hits than the second best search engine, Lycos (tables 9.2 and 9.10).

9.1 Web Servers

We have three input files that look separately for the Apache Web Server, for the Microsoft Internet Information Server and for the Microsoft Personal Web Server. Table 9.1 shows, that these input files generate most of the results (in average 1050 hits for Apache, 2500 for MS-IIS and 250 for

AltaVista::Advanced	June 13	June 14	June 15	June 16	June 19
Access.conf	4	4	1	4	4
AltaVista	141	154	153	153	167
Apache	442	442	442	446	442
Log dirs	217	217	217	217	217
ASP pages	109	117	112	112	110
Cart32	219	219	219	219	219
Count.cgi	59	253	253	253	253
Glimpse	224	224	224	224	224
htaccess	81	141	122	141	141
httpd.conf	15	12	5	12	17
inetd.conf	0	4	4	7	7
var/log/messages	3	3	3	3	3
MS-IIS	606	604	594	593	596
MS-PWS	19	28	14	28	25
etc/passwd	98	221	68	221	222
.rhosts	6	6	5	5	6
sendmail.cf	5	2	5	5	5
srm.conf	0	6	0	5	5
Sum	2248	2657	2441	2648	2663

Table 9.3: Detailed analysis of the results returned by AltaVista::AdvancedWeb

EuroSeek	June 13	June 14	June 15	June 16	June 19
Access.conf	0	0	0	0	0
AltaVista	1	1	1	1	1
Apache	31	31	31	0	0
Log dirs	0	0	0	0	0
ASP pages	0	0	0	0	0
Cart32	12	12	12	0	0
Count.cgi	0	0	0	0	0
Glimpse	105	105	105	0	0
htaccess	0	0	0	0	0
httpd.conf	0	0	0	0	0
inetd.conf	0	0	0	0	0
var/log/messages	1	1	1	0	0
MS-IIS	49	49	49	0	0
MS-PWS	0	0	0	0	0
etc/passwd	0	0	0	0	0
.rhosts	0	0	0	0	0
sendmail.cf	0	0	0	0	0
srm.conf	0	0	0	0	0
Sum	199	199	199	1	1

Table 9.4: Detailed analysis of the results returned by EuroSeek

Excite	June 13	June 14	June 15	June 16	June 19
Access.conf	0	0	0	0	0
AltaVista	1	1	1	1	1
Apache	0	0	0	0	0
Log dirs	0	0	0	0	0
ASP pages	0	0	0	0	0
Cart32	0	0	0	0	0
Count.cgi	0	0	0	0	0
Glimpse	0	0	0	0	0
htaccess	0	0	0	0	0
httpd.conf	0	0	0	0	0
inetd.conf	0	0	0	0	0
var/log/messages	0	0	0	0	0
MS-IIS	0	0	0	0	0
MS-PWS	0	0	0	0	0
etc/passwd	0	0	0	0	0
.rhosts	0	0	0	0	0
sendmail.cf	0	0	0	0	0
srm.conf	0	0	0	0	0
Sum	1	1	1	1	1

Table 9.5: Detailed analysis of the results returned by Excite

GoTo	June 13	June 14	June 15	June 16	June 19
Access.conf	0	0	0	0	0
AltaVista	0	0	0	0	0
Apache	0	0	0	0	0
Log dirs	0	0	0	0	0
ASP pages	0	0	0	0	0
Cart32	0	0	0	0	0
Count.cgi	0	0	0	0	0
Glimpse	0	0	0	0	0
htaccess	0	0	0	0	0
httpd.conf	0	0	0	0	0
inetd.conf	0	0	0	0	0
var/log/messages	0	0	0	0	0
MS-IIS	0	0	0	0	0
MS-PWS	0	0	0	0	0
etc/passwd	0	0	0	0	0
.rhosts	0	0	0	0	0
sendmail.cf	0	0	0	0	0
srm.conf	0	0	0	0	0
Sum	0	0	0	0	0

Table 9.6: Detailed analysis of the results returned by GoTo

Google	June 13	June 14	June 15	June 16	June 19
Access.conf	0	0	0	0	0
AltaVista	30	30	30	30	30
Apache	8	8	9	10	8
Log dirs	0	0	0	0	0
ASP pages	82	81	83	81	84
Cart32	19	19	19	19	20
Count.cgi	0	0	0	0	0
Glimpse	190	190	190	190	190
htaccess	0	0	0	0	0
httpd.conf	0	0	0	0	0
inetd.conf	3	3	3	3	3
var/log/messages	0	0	0	0	0
MS-IIS	9	9	9	9	9
MS-PWS	178	174	178	178	178
etc/passwd	0	0	0	0	0
.rhosts	0	0	0	0	0
sendmail.cf	0	0	0	0	0
srm.conf	0	0	0	0	0
Sum	519	514	521	520	522

Table 9.7: Detailed analysis of the results returned by Google

HotBot	June 13	June 14	June 15	June 16	June 19
Access.conf	0	0	0	0	0
AltaVista	0	0	0	0	0
Apache	0	0	0	0	0
Log dirs	0	0	0	0	0
ASP pages	0	0	0	0	0
Cart32	0	0	0	0	0
Count.cgi	0	0	0	0	0
Glimpse	0	0	0	0	0
htaccess	0	0	0	0	0
httpd.conf	0	0	0	0	0
inetd.conf	0	0	0	0	0
var/log/messages	0	0	0	0	0
MS-IIS	0	0	0	0	0
MS-PWS	0	0	0	0	0
etc/passwd	0	0	0	0	0
.rhosts	0	0	0	0	0
sendmail.cf	0	0	0	0	0
srm.conf	0	0	0	0	0
Sum	0	0	0	0	0

Table 9.8: Detailed analysis of the results returned by HotBot

Infoseek	June 13	June 14	June 15	June 16	June 19
Access.conf	0	0	0	0	0
AltaVista	49	49	49	49	49
Apache	1	1	1	1	1
Log dirs	300	300	300	300	300
ASP pages	0	0	0	0	0
Cart32	41	41	41	41	41
Count.cgi	0	0	0	0	0
Glimpse	1	1	1	1	1
htaccess	0	0	0	0	0
httpd.conf	36	36	36	36	36
inetd.conf	1	1	1	1	1
var/log/messages	0	0	0	0	0
MS-IIS	923	923	923	923	923
MS-PWS	32	32	32	32	32
etc/passwd	0	0	0	0	0
.rhosts	0	0	0	0	0
sendmail.cf	0	0	0	0	0
srm.conf	0	0	0	0	0
Sum	1384	1384	1384	1384	1384

Table 9.9: Detailed analysis of the results returned by Infoseek

Lycos	June 13	June 14	June 15	June 16	June 19
Access.conf	3	3	3	3	3
AltaVista	103	150	152	145	155
Apache	252	250	254	262	248
Log dirs	69	62	66	64	58
ASP pages	42	43	43	43	44
Cart32	296	259	109	108	299
Count.cgi	0	0	0	0	0
Glimpse	251	446	447	275	436
htaccess	16	16	16	15	16
httpd.conf	25	27	27	26	27
inetd.conf	2	2	2	2	2
var/log/messages	1	2	2	2	2
MS-IIS	278	282	285	296	284
MS-PWS	10	12	10	10	12
etc/passwd	0	0	0	0	0
.rhosts	0	0	0	0	0
sendmail.cf	0	0	0	0	0
srm.conf	1	1	1	1	1
Sum	1349	1555	1417	1252	1587

Table 9.10: Detailed analysis of the results returned by Lycos

NorthernLight	June 13	June 14	June 15	June 16	June 19
Access.conf	0	0	0	0	0
AltaVista	58	55	56	57	57
Apache	8	8	8	8	8
Log dirs	0	0	0	0	0
ASP pages	31	31	31	31	28
Cart32	1	1	1	1	1
Count.cgi	300	300	300	300	300
Glimpse	300	300	300	300	300
htaccess	0	0	0	0	0
httpd.conf	0	0	0	0	0
inetd.conf	0	0	0	0	0
var/log/messages	3	3	3	3	3
MS-IIS	373	356	356	373	382
MS-PWS	7	7	7	7	7
etc/passwd	0	0	0	0	0
.rhosts	0	0	0	0	0
sendmail.cf	0	0	0	0	0
srm.conf	0	0	0	0	0
Sum	1081	1061	1062	1080	1086

Table 9.11: Detailed analysis of the results returned by NorthernLight

MS-PWS). A fact for all attacks is however, that not all results returned by Cercator refer to the standard documentation. Even though they match our queries and filters, they do not concern what we are searching. As there are so many results, it was not possible for us to estimate the part of bad results.

Concerning the valid sites (i.e. those with the standard documentation), one cannot say, that they are necessarily vulnerable. It is likely, that several of these sites ignore that they are publishing the standard documentation of the web server they are using. So they are misconfigured sites, but that might not be the case. A site administrator can also deliberately publish the documentation without being in charge of a vulnerable site. The other problem is that we do not know the version of the server. The documentation sometimes gives indications, but it might be, that the server was updated or patched in the meantime whereas the documentation remained unchanged.

After these remarks it follows, that the numerous results do not enable us to make any firm affirmation concerning the security of the referenced site. One can only say, that it is strange, that certain sites publish the standard web server documentation.

Let us finish with showing the URLs of several examples that seemed particularly interesting in results 9.1 (partially sanitised for obvious reasons).

```
★ Rank: 825
  Url: http://xxxxxx.xxx.navy.mil/manual/
  Title: Apache 1.3 documentation
  Description: Apache HTTP Server Version 1.3. Apache 1.3 User's Guide. Release Notes.
  New features in Apache 1.3. Upgrading to Apache 1.3. Apache License. Apache...
  Date: 7-Oct-1998
  Size:
  Found by: AltaVista::AdvancedWeb

★ Rank: 360
  Url: http://xxxxxxxxx.xxxx.xxx.edu/samples/default.htm
  Title: Microsoft Internet Information Server
  Description: The Web Server Designed For Windows NT Server. Why not add the latest features to
  your Internet Information Server? Microsoft Internet Information...
  Date: 14-Oct-1996
  Size:
  Found by: AltaVista::AdvancedWeb AltaVista

★ Rank: 300
  Url: http://xxxxx.xxx.xxx.edu/IISSamples/Default/welcome.htm
  Title: Welcome To Microsoft Personal Web Server
  Description: Welcome to Microsoft Personal Web Server 4.0. This home page is hosted on
  Microsoft Personal Web Server (PWS) 4.0. PWS turns any computer running...
  Date: 21-Apr-2000
  Size:
  Found by: AltaVista AltaVista::AdvancedWeb

★ Rank: 260
  Url: http://xxxxxxxx.xxxxx.lu/iisadmin/
  Title: Internet Service Manager
  Description: A text-decoration: none Internet Service Manager Internet Service Manager for
  Internet Information Server 3.0 Introduction WWW FTP Gopher Documentation Welcome to the
  Microsoft Internet Information Server (IIS) ...
  Date: 15 Oct 1998
  Size: 2969
  Found by: Infoseek

★ Rank: 255
  Url: http://xxxx.xxxxx.army.mil/samples/default.htm
  Title: Microsoft Internet Information Server
  Description: The Web Server Designed For Windows NT Server Why not add the latest features to
  your Internet Information Server? Microsoft Internet Information Server (IIS) makes it easier
  to do business ...
  Date: 19 May 1999
  Size: 4300
  Found by: Infoseek
```

Results 9.1: Some examples related to the Web server documentation

9.2 Web server config files

The web server related files access.conf, htaccess, httpd.conf and srm.conf are quite frequent. We got 7 hits for access.conf, 150 for htaccess, 70 for httpd.conf and 6 for srm.conf (table 9.1). Their

presence alone does not help us very much, because we cannot precisely deduce the web server used as it is used at least by Apache and NCSA httpd. These files become interesting when we can read them.

In this case there are two conclusions to take: in all cases, the remote sites suffer from a configuration error, because none of these files should be visible from the web. But only the fact, that those configuration files have been published, does not help a hacker. The malicious user has to access the files to read them because the few lines of description returned from the search engines do not carry sufficient information. In that case he lets some traces in the log files and an Intrusion Detection System can launch an alarm.

Results 9.2 shows an excerpt of the list returned by Cercator (partially sanitised for obvious reasons). All entries concern configuration files lying in directories that seem valid. The description suggests, that the files have the right structure, but it does not contain any valuable information for us. So one can only say, that all results have to be checked manually whether they are correct or not. The information got back from the search engines is too scarce to learn something about security holes. The only firm conclusion is that all those sites are badly configured and should not publish their real configuration files in use on the world wide web.

```
★ Rank: 195
  Url: http://xxx.xxxxxxxx.be/intranet/htaccess
  Title: No Title
  Description: order allow, deny allow from .xxxxxxx.be allow xxx.xx.53.110 allow xxx.xx.53.109
  deny from all ...
  Date: 5-Oct-1997
  Size:
  Found by: AltaVista AltaVista::AdvancedWeb

★ Rank: 165
  Url: http://xxx.xxxxxxxxxxxxxx.gouv.fr/htaccess
  Title: No Title
  Description: AuthUserFile /home/httpd/education/da/syste /.htpasswd AuthGroupFile
  /home/httpd/education/da/syste /.htgroupmembers AuthName Education AuthType...
  Date: 4-Jun-1998
  Size:
  Found by: AltaVista AltaVista::AdvancedWeb

★ Rank: 95
  Url: http://xxx.xxxxxx.edu/conf/httpd.conf
  Title:
  Description:  # This is the main server configuration file. See URL http://www.apache.org/
  # for ...
  Date: 31 Aug 1997
  Size: 6656
  Found by: Infoseek

★ Rank: 90
  Url: http://xxxxxxxxxxx.xxxx.edu/conf/httpd.conf
  Title:
  Description: ServerType standalone Port 80 User nobody Group #-2 ServerAdmin helpdesk@s...
  Date: 8 Aug 1996
  Size: 409
  Found by: Infoseek

★ Rank: 35
  Url: http://xxxxxxxxxx.xx.net/conf/srm.conf
  Title: http://xxxxxxxxxx.xx.net/conf/srm.conf
  Description: With this document, you define the name space that users see of your # http
  # server. # See the tutorials at http://hoohoo.ncsa.uiuc.edu/docs/tutorials/ for # more
  information. #
  Date:
  Size:
  Found by: Lycos
```

Results 9.2: Some examples related to the Web server config files

9.3 Log directories

Cercator returns many results for this search (in average 580). The results look quite valid. There are always the directories /logs/ or /_admin/logs/ with the files access_log, error_log and /httpd-access_log. As in the previous case, the results and the description field do not give us immediate reasons to tell something about the vulnerability of the respective sites. To do that, one would

have to access the files. But we can say, that the site is misconfigured because there is no reason to publish a log file directory. Results 9.3 shows a little excerpt of what Cercator found.

```
★ Rank: 90
  Url: http://xxxxxx.xxx.noaa.gov/logs/access_log
  Title: No Title
  Description: test - - [13/Apr/1999:15:19:17 -0600] "GET /xxxxxxxxxx/ HTTP/1.0" 200 489 test
  - - [13/Apr/1999:15:19:23 -0600] "GET...
  Date: 27-Jan-2000
  Size:
  Found by: AltaVista::AdvancedWeb

★ Rank: 75
  Url: http://xxxxxxxx.xxxxx.ch/logs/access_log
  Title: No Title
  Description: xxxxxxxx.xxxxx.ch - - [30/Apr/1998:13:18:31 +0200] "GET / HTTP/1.0" 200 1032
  xxxxxxxx.xxxxx.ch - - [30/Apr/1998:13:18:31 +0200] "GET /icons/menu.xbm ...
  Date: 26-Jan-2000
  Size:
  Found by: AltaVista::AdvancedWeb

★ Rank: 50
  Url: http://xxx.xxxxxxxxxx.com/logs/error_log
  Title: No Title
  Description: Mon Jan 3 21:32:30 2000] access to
  /usr/local/etc/httpd/htdocs/xxx.xxxxxxxxxx.com/cgi-bin  failed for xxx.xxx.237.65, reason:
  attempt to invoke...
  Date: 26-Jan-2000
  Size:
  Found by: AltaVista::AdvancedWeb

★ Rank: 150
  Url: http://xxx.xxx.xx.xx/linux/diald/var/log/messages
  Title: No Title
  Description: May 21 07:52:04 Anne syslogd 1.3-3#24: restart. May 21 07:52:04 Anne kernel:
  Loaded 5635 symbols from /boot/System.map-2.0.33. May 21 07:52:04 Anne...
  Date: 21-May-1998
  Size:
  Found by: AltaVista::AdvancedWeb AltaVista

★ Rank: 120
  Url: http://xxx.xxxxxxx.nl/_admin/logs/access_log
  Title: No Title
  Description: xxxxxxx.xxxx.xxxxxxxxx.nl - - [16/Oct/1999:15:31:04 +0200] "GET / HTTP/1.1" 200
  580 xxxxxxx.xxxx.xxxxxxxxx.nl - - [16/Oct/1999:15:31:08 +0200] "GET...
  Date: 27-Jan-2000
  Size:
  Found by: AltaVista::AdvancedWeb
```

Results 9.3: Some examples related to the Web server log files

9.4 Count.cgi, Glimpse

The searches for two common CGI programs returned around 550 results for the count program and approximately 1000 results for the webglimpse program. This time we are certainly not in a case of a misconfiguration error. These files are present deliberately. A valid hit in the results file, generated by Cercator tells us, that the remote site uses the count.cgi (resp. the glimpse) program. The problem for us is now to find out which version of the program is used, in order to decide whether the site is vulnerable or not. Here, one can tolerate to contact to the site because these files are anyway available for public access and no IDS system will generate an alarm. Once there is a certainty about the version, we know if the site is vulnerable or not.

For count.cgi it is relatively easy to find out the version. First, there is a big chance, that the hit returned by Cercator is already valid, because we added a date restriction. The other solution is to access the site and look for a line saying "Output generated by MKStats, version 2.3". That could be done with the special feature where Cercator can access the files and apply a filter over it. In any case, each time, we find an indication for version 2.3, we know that the site is using a vulnerable version.

For Webglimpse, we got very interesting results as well and they are all valid. Results 9.4 shows a little excerpt of results. As a conclusion one can say, that for these two tests, we found a lot of valid sites using programs that are known to have a vulnerability.

★ Rank: 260
Url: http://xxx.xxxxx.gov/xxx/ghindex.html
Title: Full search on archive xxx
Description: 99 - Articles & General info: Search. Search the full archive: xxx. String to search for: Case sensitive Partial match Jump to line misspellings allowed. Return only files modified within the... 11/18/1999
Date:
Size:
Found by: NorthernLight

★ Rank: 50
Url: http://xxxxx.xxx.nasa.gov/cgi-bin/aero/aero-wais.pl
Title: Full search
Description: WebGlimpse Search. Search on the entire archive. String to search for: Case sensitive Partial match Jump to line 0. 1. 2....
Date: 13-Apr-2000
Size:
Found by: AltaVista

★ Rank: 30
Url: http://xxx.xxxxxxxxxx.fr/wgindex.html
Title: Full search on archive WebGlimpse
Description: WebGlimpse Search Rechercher dans l'archive complete : WebGlimpse Mot rechercher : Respecter la casse 0 1 2 Fautes de frappe autorises Nombre maximum de pages retournees : 10 ...
Date: 1-Mar-2000
Size:
Found by: Google AltaVista Lycos

★ Rank: 100
Url: http://xxx.xxxxx.net/mkstats/_dkassemblies_index_html.html
Title: /dkassemblies/index.html
Description: dkassemblies/index.html. Total Hits: 253 Total Bytes: 718464 Most popular Day: 08/15/98 Last 14 Days: 10/12/98 3...
Date: 31-Oct-1998
Size:
Found by: AltaVista::AdvancedWeb

★ Rank: 50
Url: http://xxx.xxxxx.com.au/mkstats/browser_idx.html
Title: Browser Links
Description: 52 - Articles & General info: Tuesday Oct 15, 1996 Browser Links Not implemented Output generated by MKStats 10/27/1997
Date: 27-Oct-1997
Size:
Found by: NorthernLight AltaVista::AdvancedWeb

Results 9.4: Some examples related to the CGI attacks

9.5 AltaVista Search Engine

Even though, our strategy for finding sites using the AltaVista software to implement their local search engine seemed quite weak, Cercator found a lot of valid sites (about 160 out of 400). That is mostly due to the efficient filters. This time again, the fact that we have results is normal. There is no configuration error at the respective sites. The vulnerability that we try to find lies in the software being used.

Results 9.5 shows an excerpt of the 400 examples that Cercator found. From the description part returned by the search engine, we can almost always deduce, whether a hit is valid or not. The entry often contains an information about the version. When the version is 2.0b or 2.3A we know that the site suffers from the bug. If the description does not give information, we can access the site and browse around to find indications. There is no problem, when we access the files, because they are published for being seen. In our results, there are 53 entries that specify that their version is 2.3.

So this is a really good example for showing the usability of our strategy. The search engines return enough information for deciding whether a site is using a vulnerable software version or not.

```
★ Rank: 435
  Url: http://xxxxxx.xxxxx.gov/
  Title: AltaVista Search Intranet V2.3A: Simple Query xxxxx Search Utility
  Description: Search xxxxx World Wide Web for documents in any language Chinese Czech Danish
  Dutch English Estonian Finnish French German Greek Hebrew Hungarian Icelandic Italian Japanese
  Korean Latvian Lithuanian
  Date: 14-Jan-2000
  Size:
  Found by: AltaVista AltaVista::AdvancedWeb Lycos
★ Rank: 480
  Url: http://xxx.xxx.edu:9000/
  Title: AltaVista Search Intranet V2.3A: Simple Query
  Description: Search xxxxxxxxxxxxxxxxxx Help Search The Intranet World Wide Web for documents
  in any language Chinese Czech Danish Dutch English Estonian Finnish French German Greek Hebrew
  Hungarian Icelandic Ital
  Date: 11-Jan-2000
  Size:
  Found by: AltaVista AltaVista::AdvancedWeb Lycos
★ Rank: 290
  Url: http://xx.xxxxxxxxxxxxxxxxx.xxxxxxx.fr/
  Title: Moteur de recherche UREC : requete simple
  Description: Requete AltaVista Search Intranet V2.3...
  Date: 11-Jan-2000
  Size:
  Found by: AltaVista
```

Results 9.5: Some examples related to the AltaVista Search Engine

9.6 Active Server Pages

This time, we are not looking for a particular program, but for sites having ASP pages. Up to now, there is no problem, and there are plenty of such sites. But we specialise the search in order to find only wrong ASP pages. In fact non debugged active server pages can represent a serious security hole.

Cercator returns us about 330 valid sites in average. Results 9.6 shows three examples. Each entry stands for an asp page, that was not working correctly at the moment it was indexed. To exploit the problem, one has to access the site to read the source code of the include files and to gain information about the site. But, as we do not want to do that, we have no idea on how many of the results are still active and which of them have already been corrected.

We can conclude, that each time, where the link returned by the search engine is still active and pointing to the same page, then the site could potentially be facing a security problem.

```
★ Rank: 240
  Url: http://xxx.xxxxxxxxxxxxxxxx.com/default.asp
  Title: Microsoft VBScript runtime error '800a0035' File not found /boPublic.inc, line 9...
  Description: 99 - Articles & General info: font face="Arial" size=2> Microsoft VBScript runtime
  error'800a0035' File not found. boPublic.inc, line 9 Date Not Available
  Date:
  Size:
  Found by: NorthernLight
★ Rank: 195
  Url: http://xxx.xxxxxxxxxxxx.ca/prizes/ggla/default.asp
  Title: http://xxx.xxxxxxxxxxxx.ca/prizes/ggla/default.asp
  Description: Microsoft VBScript runtime error '800a000d' Type mismatch: 'cInt'
  /scripts/init-e.inc, line 11
  Date: 3-Mar-2000
  Size:
  Found by: AltaVista Lycos LycosSimple
★ Rank: 30
  Url: http://xxx.xxxxxxxxxxxxxxxxxx.co.uk/03/l.asp
  Title: No Title
  Description: Microsoft VBScript runtime error '800a004c' Path not found.
  D:/INETPUB/WWWROOT/_WEBSITES/R DIT/WWW/03/../_scripts/start_session.inc, line 39 ...
  Date: 13-Apr-2000
  Size:
  Found by: AltaVista
```

Results 9.6: Some examples related to the ASP attack

9.7 Cart32

For the cart32 problem, the 500 results returned by Cercator are always valid, as the filter only keeps hits that contain cart32.exe in the url. They are always referring to sites using the Cart32 e-business software. The description gives us often information about the version of the program in use. Therefore it becomes relatively easy for us to decide which sites are vulnerable and which are not. There is however a hypothesis behind. When the search engine gives us a link like those shown in results 9.7, we assume, that on the site pointed by the link the program is running in the version indicated. But that hypothesis is not verified all the time. The link returned by the search engine could be broken or the software could have been updated without changing the html pages.

So in conclusion, we can say that all hits are relevant and the sites are suffering a security bug, unless they patched their software after April 2000. This bug has been publicly known and a patch released by that time.

9.8 /etc/passwd

Cercator found almost 300 sites with a password file readable from the web. This is not a large number, and it makes us suppose, that search engines implement certain safety filters by their own, that prevent them from indexing certain files. A filter in the query guarantees us, that we only get /etc/passwd files. But we have no information whether the files still exist or have been removed in the meantime. Even if the files exist, we do not know whether they are the real system password files or whether they are just files called passwd and having the structure of a real password file. There is no way for us to gain certainty on this point.

A second filter increases the ranking of those results where the password is not shadowed. So these hits are shown first in the results file. When the results we got back from the search engines are really what they seem to be, the concerned sites have a serious security problem. The description part returned by the search engines contains enough information about the site that we can attack it directly without ever having contacted it before. It contains the login and the password of root. With a cracker program it is possible to decrypt the password in a couple of days.

In this context here, we do not want to access the file anymore as we could have in previous

★ Rank: 1370
Url: http://xxx.xxxxxxxxxx.com/scripts/cart32.exe/xxxxxxxx-ItemList
Title: Cart32 v2.5a
Description: Cart32 v2.5aShopping Cart System for Windows 95 NT http://www.cart32.com Registered to xxxxxxxxxxxxxxxxx client version 1996-98 McMurtrey/Whitaker Associates, Inc. Build 353
Date:
Size:
Found by: Lycos

★ Rank: 400
Url: http://xxx.xxxxxxxxxxxxxxxxxx.com/_scripts/cart32.exe/xxxxxx-Empty
Title: Cart32 v2.6
Description: Cart32 v2.6Shopping Cart System for Windows 95 NT http://www.cart32.com Registered to xxxxxxxxxxxxxxxxxxx 1 client version 1996-98 McMurtrey/Whitaker Associates, Inc. Build 443
Date:
Size:
Found by: LycosSimple Lycos

★ Rank: 335
Url: http://xxx.xxxxxxxxxxxxxxxxx.com/xxxxxxxxxxx/cgi-bin/
cart32.exe/xxxxxxxxxxx-Empty
Title: Cart32 v2.5a
Description: Cart32 v2.5a Shopping Cart System for Windows 95 & NT. http://www.cart32.com. Registered to xxxxxxxxxxxxx 1 client version. 1996-98...
Date: 12-Jan-2000
Size:
Found by: AltaVista AltaVista::AdvancedWeb

★ Rank: 255
Url: http://xxx.xxxxxxxxxxxxxxx.com/cgi-shl/cart32.exe/xxx-ItemList
Title: Cart32 v3.0
Description: Cart32 v3.0 Shopping Cart System for Windows. http://www.cart32.com. Registered to xxxxxxxxxxxxxxxxxxxxx. License: 1 client(s) 1996-99...
Date: 14-Jan-2000
Size:
Found by: AltaVista::AdvancedWeb

Results 9.7: Some examples related to the Cart32 attack

cases, because that would leave suspicious traces in a log file. So we cannot find out what the other user IDs declared in the password file nor can we read other configuration files lying in the /etc directory.

Results 9.8 shows a small excerpt of all the results found by Cercator (partially sanitised for obvious reasons). There are two points particularly notable. First, we saw, that there are 8 from the 297 entries that show the not shadowed root password! And second it is notable, that the site http://xxx.xxxxx.net comes up 69 times, but each time with a different user. xxxxx.net is an ISP that offers dial up access and Web hosting to its customers. They need the numerous /etc/passwd files for the individual ftp accesses of their customers. So the hit does not concern the real /etc/passwd file of the system, but nevertheless, the site suffers from a misconfiguration error because /etc/passwd should never become indexed by search engines nor be visible on the world wide web.

9.9 .rhosts

Cercator did not find many .rhosts files so that we can consider all the six found in detail (results 9.9). The results are each time valid. The description returned shows that the file has the right format, but it does not show the whole content of the file. What we would like to see, are .rhosts files which have a ++, but the description does not show any. Unfortunately we cannot search for that expression directly either, because the plus symbol is a reserved character for all search engines. But in any case, the results that we get (results 9.9) contain the names of certain hosts in the remote network and that is already an interesting information.

So the conclusion of this attack is, that there are six misconfigured sites. In fact the .rhosts file should never be indexed by the search engines nor be visible on the web.

★ Rank: 590
Url: http://xxxxxxx.xxx.xxxx.edu/ftp/etc/passwd
Title: No Title
Description: root:/Hk29fWlxa2a:0:0:Super-U er,,,,,,,:/:/bin/csh bin:*:2:2:System Tools
Owner:/bin:/dev/null
sys:*:4:0:System Activity Owner:/var/adm:/bin/sh...
Date: 6-Jun-1995
Size:
Found by: AltaVista::AdvancedWeb

★ Rank: 320
Url: http://xxx.xxx.edu/afs/athena/system/xxxxxxxxx/srvd/etc/passwd
Title: No Title
Description: root:hsIkR4sq3:0:1:System PRIVILEGED Account:/:/bin/csh operator:PASSWORD
HERE:0:28:Operator PRIVILEGED Account:/opr:/opr/opser...
Date: 29-Sep-1992
Size:
Found by: AltaVista::AdvancedWeb

★ Rank: 140
Url: http://xxx.xxxxx.mil/aftp/etc/passwd
Title: No Title
Description: root:*:0:1:system PRIVILEGED account:/:/bin/sh nobody:*Nologin:65534:65534:anonymous
NFS user:/:nobodyV:*Nologin:60001:60001:anonymous SystemV.4 NFS...
Date: 5-May-1995
Size:
Found by: AltaVista AltaVista::AdvancedWeb

★ Rank: 135
Url: http://xxx.xxxxx.net/users/xxxxxxx/etc/passwd
Title: No Title
Description: root:*:0:0:System Administrator:/root:/bin/csh daemon:*:1:1:System
Daemon:/:nologin uucp:*:6:6:UNIX-to-UNIX...
Date: 11-Nov-1999
Size:
Found by: AltaVista AltaVista::AdvancedWeb

★ Rank: 125
Url: http://xxx.xxxx.xxx.nist.gov/~ftp/etc/passwd
Title: No Title
Description: root:x:0:1:Super-User:/:/sbin/ csh daemon:x:1:1::/:bin:x:2:2::/usr/bin:
sys:x:3:3::/: adm:x:4:4:Admin:/var/adm: lp:x:71:8:Line Printer...
Date: 7-Sep-1999
Size:
Found by: AltaVista AltaVista::AdvancedWeb

Results 9.8: Some examples related to the passwd file

★ Rank: 145
Url: http://xxx.xx.xxx.edu/xxxxxxxx/mpi/rhosts
Title: No Title
Description: aleph aleph1 aleph2 aleph3 aleph4 aleph5 aleph6 aleph7 aleph8 aleph9 aleph10 aleph11 aleph12 aleph13 ...
Date: 30-Sep-1999
Size:
Found by: AltaVista AltaVista::AdvancedWeb

★ Rank: 145
Url: http://xxx.xxxxxxxxxx.or.jp/people/xxxxx/linux/_rhosts
Title: No Title
Description: xxxxxxxxxxx ...
Date: 27-Aug-1999
Size:
Found by: AltaVista AltaVista::AdvancedWeb

★ Rank: 125
Url: http://xxx.xxxxxxxx.or.jp/xxxxxxxx/linux/rhosts
Title: No Title
Description: koala eucalyptus kangaroo rabbit...
Date: 20-Sep-1998
Size:
Found by: AltaVista AltaVista::AdvancedWeb

★ Rank: 95
Url: http://xxxx.xxxx.xxxxx.edu.tw/u/xxxxxxx/rhosts
Title: No Title
Description: Here you need to replace joe by your own login name# host7 joe host8 joe host9 joe host10 joe host26 joe host27 joe host7.xxxx.xxxxx.edu.tw joe...
Date: 9-Oct-1998
Size:
Found by: AltaVista AltaVista::AdvancedWeb

★ Rank: 65
Url: http://xxx.xxx.xx.edu/xxxxxxxxxxxxx/current/part1/rhosts
Title: No Title
Description: xxxx.xxxxxx.xx.edu YOUR_AFS_ID xxxxxx.xxxxxx.xx.edu YOUR_AFS_ID xxxxxxx.xxxxxx.xx.edu YOUR_AFS_ID xxxxxxx.xxxxxx.xx.edu YOUR_AFS_ID...
Date: 17-Apr-1999
Size:
Found by: AltaVista AltaVista::AdvancedWeb

Results 9.9: All results for the .rhosts attack

9.10 sendmail.cf

Here we are again in a case, where the presence of the file alone reveals already the existence of a specific program. But each valid hit is due to a misconfiguration. The usability of this is however restricted. As there is no way to discover the version of sendmail being used, we cannot say anything on the vulnerability of the site. Concerning the content of the sendmail.cf file, it is of limited interest. Even if we were allowed to read it, it would only give indirect information about the site, but present no real opportunity for attacking.

In our concrete case, we got five results (Results 9.10) from AltaVista. Only three results concern sendmail.cf files and one cannot say for sure, that they are all valid, because the lie in very untypical directories for a real sendmail.cf file. One result concerns a mirroring site for sendmail.cf and the last one concerns a CGI program.

So as a conclusion, one can say, that for sendmail.cf we did not find any interesting results.

★ Rank: 150
Url: http://xxx.xxxxxxxxx.ru/pub/FreeBSD/FreeBSD-current/src/contrib/sendmail/cf
Title: xxxxxxxxx.xxxxxx.ru:/pub/FreeBSD/FreeBSD-current/src/contrib/sendmail/cf
Description: pub/FreeBSD/FreeBSD-current/sr /contrib/sendmail/cf. <Up> <dir> Back to the parent directory cf <dir> domain <dir> feature <dir> hack <dir> m4 <dir>...
Date: 21-Feb-2000
Size:
Found by: AltaVista AltaVista::AdvancedWeb

★ Rank: 140
Url: http://xxx.xxxxxxxxx.com/cgi-bin/cvsweb.cgi/mail/sendmail/cf
Title: Plug & Play Linux GroundZero Development Tree: /mail/sendmail/cf
Description: xxxxxxxxx Plug & Play Linux GroundZero Development Tree. Click on a directory to enter that directory. Click on a file to display its revision...
Date: 29-Feb-2000
Size:
Found by: AltaVista AltaVista::AdvancedWeb

★ Rank: 140
Url: http://www.xxx.xxxxxxxx.es/xxxxxxxx/_Utopia/sendmail.cf
Title: No Title
Description: Copyright (c) 1983 Eric P. xxxxxxxx # Copyright (c) 1988, 1993 # The Regents of the University of xxxxxxxx. All rights reserved. # # Redistribution...
Date: 6-May-1996
Size:
Found by: AltaVista AltaVista::AdvancedWeb

★ Rank: 100
Url: http://xxxxxxx.xxx.xx.ac.uk/Mirrors/xxxxxxxx/pub/freeware/sendmail.cf
Title: No Title
Description: SENDMAIL CONFIGURATION FILE FOR SUBSIDIARY MACHINES # # You should install this file as /etc/sendmail.cf # if your machine is a subsidiary machine...
Date: 4-Jun-1997
Size:
Found by: AltaVista AltaVista::AdvancedWeb

★ Rank: 90
Url: http://xxxxxx.xxx.xx.ac.uk/xxxxxx/xxxxxx.cgi?sendmail.cf
Title: sendmail.cf from xxxxxx
Description: sendmail.cf. <messaging> sendmail's configuration file, which it reads once when starting up, usually found in the /etc directory. Only real Unix...
Date: 11-Feb-2000
Size:
Found by: AltaVista AltaVista::AdvancedWeb

Results 9.10: All results for the sendmail.cf file

Chapter 10

Conclusion

This project presented an analysis of how one can use the common search engines available on the Internet to make very precise investigations. We found out, that each search engine has its own characteristics and its own strengths. In order to make good queries, it is useful to contact more than one search engine. That gives us a larger number of hits and permits us to better validate some of them: when several search engines find the same hit, it may indicate that the result is particularly well matching the query and so should be given a higher ranking. That are two reasons for using more than one search engine. Another result of this analysis was, that we found it useful to check the numerous results retrieved from all search engines before showing them to the user, as they are by far not always what one is looking for. As we did not find a metacrawler on the Internet that met these conditions, we decided to create a new metacrawler that would allow to make precise queries using the whole range of options offered by the search engines. The second particularity of the new metacrawler is its possibility to refine the results delivered by the search engines before delivering them to the user.

So one result of the project is a new metacrawler – Cercator– that is designed in a very general way in order to allow all sorts of searches on the Internet. It contacts several search engines and can filter the results it gets back. Afterwards, in order to verify the functioning of that tool, we used it for a special topic: we used it to search for computers with security holes on the Internet.

As first general conclusion one can say, that it is possible to gain sensible information about remote computer systems by only asking the search engines. However, that information is rarely sufficient to declare that a site is certainly vulnerable to a specific attack. We found examples, where the information was complete enough or where it could be completed by the user, and other examples that allowed only suspicions. One has to keep in mind, that search engines do not exist to tell us all the sensible information we would like to know about remote sites. So the challenge is to write clever queries in order to get as much valid information from the search engines as possible. Even if there is only a portion of classical attacks where search engines can give hints, we have to keep in mind though, that one hole is enough to take control over a remote machine. Our inability to carry out an exhaustive search of all vulnerabilities is not an issue here. Search engines allow to search for specific files. The existence of the files can suggest either a bad configuration because the files should not be available or simply the installation of a certain program. The main obstacle to more straight results is the self-imposed interdiction to access the remote site and look at the content of the critical files.

Even though the main creative and manual work lies in the creation of the input files, the results obtained from Cercator have to be interpreted manually again. Normally the results give us indications about sites, that seem to be misconfigured. The results can give ideas to a hacker where and how to start an attack, but the bare results give rarely a guarantee of success. Of course that depends a lot on what one is currently looking for. Both cgi examples, the AltaVista, ASP, cart32 and partially the /etc/passwd examples are working well. In these cases the chance

is really high, that the vulnerability can be directly exploited to launch a successful attack. A general problem is the freshness of information we got from the search engines. We are completely relying on the information they give us. So if we think, that a site may suffer from a certain bug, because the search engine returned us a particular link, that does not mean, that the link is still valid and that the problem has not yet been corrected. During our tests, we noticed that a considerable portion of the links were no more active.

So as a second conclusion one can say, that the strategy and the tool are useful but can be expensive in manual efforts in relation to the successes that can be expected. It suits better for someone always having plenty of time and wanting to hack a site for fun. The results give him indications of systems he could succeed to hack. But to do that, he has to contact the remote site to gain the supplemental information he needs.

So the good solution – that we recommend – is to continue to write more input files and more back ends for new search engines, in particular for private search engines. Then one runs Cercator once a month with all the input files and with the best search engines as parameter. After that, one runs a filter-program (that we have written as well) with the results from Cercator. The filter retains only the sites of the customers one wants to protect. There will normally not be too much results left, so that it becomes possible to analyse them manually and in detail. Perhaps the customer allows to access its site in order to test its IDS system. Then we can take the role of the hacker and look precisely at what files are readable and how sensitive is their content.

And as a last conclusion we can say, that Cercator has proven its efficiency when we used it to search for security holes. It turns out that it is a very general program, that can be useful in many other occasions as those in this context. It is a very powerful metacrawler that accepts complex queries and delivers them to several search engines at once. Its real strength lies in the possibility to refine the results in a series of complex filters. In chapter 8 we gave several examples. At the moment its performance is not very high yet, but we have presented a concept in section 6.3 that could improve that problem.

Chapter 11

Acknowledgements

At the end of this project, I wish to thank all those that helped me to conduct it towards a good end. My first thanks go to Prof. Refik Molva – my Eurecom supervisor – who helped me to get this interesting project and who guided me during the six months. Then comes my company supervisor Dr. Marc Dacier. He let me a lot of freedom with the project. Every time he came back from one of his journeys, he asked me about the current state of the project and gave me indications on the way to continue. The third person I wish to thank is Lt. Charles-Emmanuel Daviet from the Ecole spéciale militaire of Saint Cyr in France. He spent a short internship here at the IBM research Lab. in Zurich and helped me with my project. Charles-Emmanuel wrote the Graphical User Interface to create the XML files and he participated in the development of the strategies that allowed to find the vulnerabilities. Later he coded these strategies in the input files. I wish to mention Candid Wüest and Andreas Tscharner who taught me a lot of background information about vulnerabilities and hacking. My last thanks go to Christian Gigandet and Céline Steenkeste who I had the great chance to share the office with. They often gave me some good advises for my project about Perl or XML and they created an atmosphere in the office that made the whole internship at IBM a great pleasure.

References

[1] *How search engines can be used to locate millions of vulnerable web sites.* TISC Insight Newsletter, Feb 2000. http://tisc.corecom.com/newsletters/24.html.

[2] *Using search engines to locate millions of vulnerable web applications*, Feb 2000. http://www.perfectotech.com/blackwatchlabs/vul2_17.htm.

[3] Holger Reibold and Franz Neumeier. *Findigkeit gefragt.* PC Professionell, page 170, Feb 2000. Ziff Davis.

[4] http://www.searchengines.com/.

[5] *WWW Search Engines*, 1999. http://www.lib.utk.edu/refs/search.html.

[6] *Index of Internet search engines and web directories*, 2000. http://www.allsearchengines.com/.

[7] *Search the World Wide Web*, Mar 2000. http://www.murdoch.edu.au/srch/search.html.

[8] Danny Sullivan. *Search Engine Watch.* http://www.searchenginewatch.com/.

[9] Martijn Koster. *Robots in the Web: threat or treat ?* ConneXions, Vol 9, Apr 1995. http://info.webcrawler.com/mak/projects/robots/threat-or-treat.html.

[10] Martijn Koster. *A Standard for Robot Exclusion.* http://info.webcrawler.com/mak/projects/robots/norobots.html, 1994.

[11] Andrew Rigby. *BIG Search Engine Index*, 2000. http://www.search-engine-index.co.uk/.

[12] Klaus Schallhorn. *Die Suchmaschinen Toolbox*, May 2000. http://www.kso.co.uk/.

[13] Dirk van Eylen. *Ranking of search results using Altavista*, 1997. http://www.ping.be/dirk_van_eylen/avrank.html.

[14] Tim Bray, Jean Paoli and C. M. Sperberg-McQueen. *Extensible Markup Language (XML) 1.0*, Feb 1998. http://www.w3.org/TR/1998/REC-xml-19980210.

[15] Norman Walsh. *What Do XML Documents Look Like ?*, Oct 1998. http://www.xml.com/pub/98/10/guide2.html.

[16] *Common Vulnerabilities and Exposures (CVE).* http://cve.mitre.org.

[17] *Securityfocus.* http://www.securityfocus.com.

[18] Eric Armstrong. *Creating a Document Type Definition.* http://java.sun.com/xml/docs/tutorial/sax/5a_dtd.html.

[19] Simon Cozens. *What's New in Perl 5.6.0 ?*, Apr 2000. http://www.perl.com/pub/2000/04/whatsnew.html.

[20] Lt. Charles-Emmanuel Daviet. *Détection de vulnérabilités par un méta-moteur de recherche*, May 2000. Ecole spéciale militaire de Saint Cyr, Internal Report.

[21] Jerry Walsh. *Security alert — using includes improperly from non-debugged asp pages can allow visitors to view your souce code.* http://www.4guysfromrolla.com/webtech/020400-2.shtml.

[22] Muhammad Muquit. WWW Homepage Access Counter and Clock, 2000. http://www.muquit.com/muquit/software/Count/Count.html.