ADITYA K SOOD A.K.A
0KN0CK

Hacking Through Wild Cards

Difficulty

▨ ☐ ☐

This paper sheds light on the usage of wild characters that lead to hacking. The wild characters are used effectively in a different sphere. The inappropriate use of wild characters can lead to misconfiguration of parameters thereby resulting in a number of attacks.

Many authentication bypass vulnerabilities occur due to improper use of wild cards. The set of characters can be used tactically to fingerprint running software such as web servers. The Meta characters can be fused with HTTP verbs to query the version of remote web servers and the way different servers react to requests fused with Meta characters can be observed (there is something missing here so I have added *can be observed*). A misconfigured zone configuration file, due to wild cards, can impact the DNS on large scale. Even search queries are dependent extensively on these set of characters where they act as a prime point of search engine hacking. The core aim is to understand the paradigm of wild and meta character functionality and its stringent usage that results in building of an attack surface. The paper will cover different types of attacks and hacking entities related to Wild characters. I will be using wild cards and wild characters terms interchangeably.

Explanation

The use of wild characters plays a critical role in making things plausible as well as problematic. It depends a lot on the context in which it is applied. The context here refers to the implementation. The right approach gives very specific outcomes while the wrong implementation can jeopardize normal operation. On the contrary, wild characters can also be used for testing purposes. This issue will be proven with an example of testing web server responses to grab banners. The responses from different web servers are always in variation. The wild cards can be used to launch different types of attacks when certain conditions are met. For Example: – a pure denial of service attack at an application level in a three tier architecture. Of course, one can not ignore the interim behavior of wild cards in a search engine. The wild card characters can be used in a crafty manner by penetration testers and hackers to search and explore the hidden entities that leverage vulnerability patterns on the web. For Example: – vulnerability finding through a search engine like Google. The Google hacking database is a perfect example of this. Even a specific wild card is used in DNS names to resolve the domain structures between primary and secondary sub domains etc. The XSS level attacks whether persistent or reflective are some what triggered by wild cards too. We will also be covering administrative issues because the inappropriate presence of a single wild character can subvert the functionality of the Internet.

We will be discussing the impact of wild characters in different areas of computer security by discussing some cases.

DNS Behavior – (*) Wild Card Stringency

The wild card plays a critical role in differentiating between the domain and sub domains. The

WHAT YOU SHOULD KNOW...

Basic behavior of Wild Cards

Logic Creation using Wild Cards

WHAT YOU WILL LEARN...

The impact of Wild Cards on security

Wild Card based Configuration Management

Generation of attack surface due to Wild Card Insecure Usage specification of the wild character in a zone configuration file is a serious concern because it can impact the network functionality on a very large scale if not implemented appropriately. The wild cards are used in the DNS configuration to match a specific sub domain or any resource record. The DNS resolving is based on the request sent by a client in the form of a query. The query parameters are mentioned below:

- Query Type
- Query Class
- Query Name

The DNS server returns a resource record after execution of the query. The mechanism of producing DNS results depends on the use of the query parameters. The record containing data is sent data back to the sender if all three query parameters are matched with the record i.e. a successful operation.

If only query name and query class is determined, but not query type then it becomes hard to extract data as DNS is unable to load data based on the name. In order to avoid the failure, the `(*)` wild character is used.

This results in more complexity, when a query class is matched but not the query name. In that case the wild card entry is treated as an answer which matches the desired domain as per the request. Let's say if a zone file is having an entry as stated below:

```
*.domain.com.    3600     MX    10
                            ret.example.com.
```

For example: – if a request is issued for temp.domain.com and it does not exist.

The presence of a wild character changes the query check procedure. The query for temp.domain.com will be matched to **.domain.com* and the DNS is resolved for *ret.example.com*. As we are talking about the MX record in the example, the MX record will be resolved to ret.example.com (once again a bit of confusion here as to what is meant). This functionality is stated in RFC 1034 which defines an issue as:

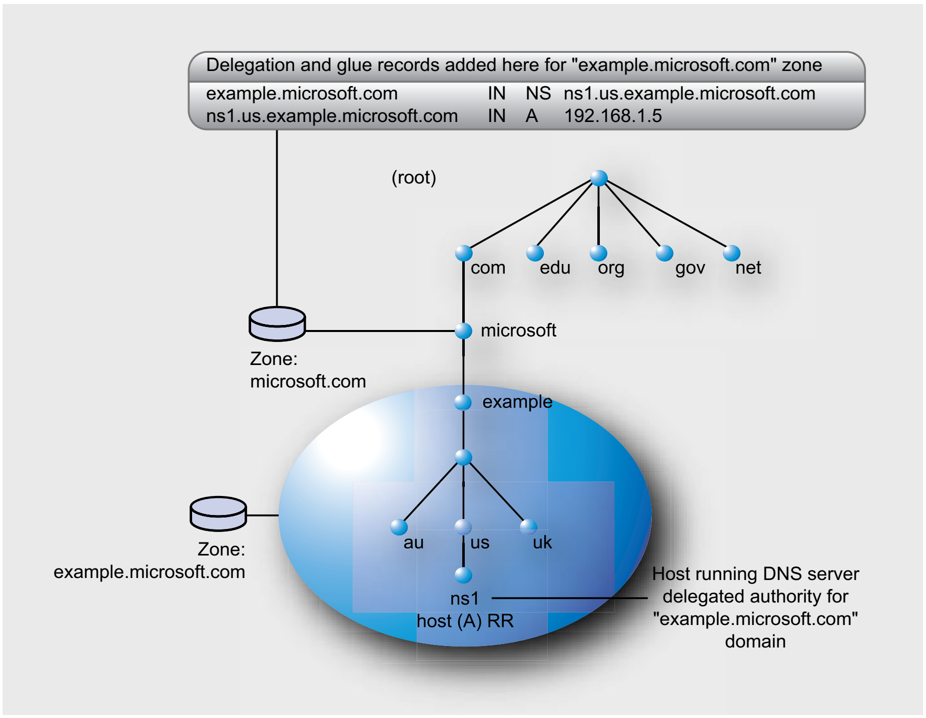
If the "*" label does exist, match RRs at that node against QTYPE. If any match, copy them into the answer section, but set the owner of the RR to be QNAME, and not the node with the "*" label

The `(*)` wild card is mentioned as the least significant (left) part when an entry has to be made in the zone file. It depends a lot on the naming convention which is used for different protocols. The naming convention defines the structure of the resource record as a DNS entry. The naming scheme is a part of DNS protocol and wild cards have a direct relation with it. Structuring of the DNS record depends a lot on the record definition. It covers:

- Explicit definition of DNS records (MX, SRV etc)
- Wild Card usage in defining DNS records (MX, SRV etc)

It criticality depends on the configuration of DNS Zone file. Records like Aaa.bbb.domain.com, Temp.ret.domain.com match a single set of records if a wild card is defined as seen in the example above. The reason for this is that DNS works as per the configuration and the resource record is mapped to the wild card character by using the standard naming scheme. Due to this, the response of the query ends up containing the same address as a resolved address. This is not at all true in the DNS context and hence creates a certain set of problems due to the existence of a single wild character.

Another problem comes into play (or *the picture*, I think *play* is better suited) if certain service specific records are present. The service records are referred to SRV records here including mail, ntp etc. These records require a protocol and port number to connect to. If we consider the aforementioned scenario, the DNS will again resolve a query on the wild character and naming scheme used in the DNS configuration. Hence the records returned as per the zone configuration will be different and it becomes hard for the sender (what sender ?) to use the records to connect to the service. It again depends a lot on the explicit and implicit definition. But we can not ignore the problem due to the fact that wild cards within DNS is used across different organizations for communication purposes. That is why the issue is so critical. We can not leverage this issue by saying it is okay within a single organization but it has a diversified impact. Certain records don't have a problem like MX (Mail). The delegation process is a very crucial part of DNS functionality. Let's have a look at the Microsoft example of DNS (see Figure 1).



Delegation and glue records added here for "example.microsoft.com" zone
example.microsoft.com IN NS ns1.us.example.microsoft.com
ns1.us.example.microsoft.com IN A 192.168.1.5
(root)
com
edu
org
gov
net
microsoft
Zone: microsoft.com
example
au
us
uk
Zone: example.microsoft.com
ns1
host (A) RR
Host running DNS server delegated authority for "example.microsoft.com" domain

Figure 1. *Microsoft – DNS Delegation*

This depends a lot on the delegation which covers:

- Crossing organization boundaries for DNS resolving i.e. Zone Transfer.
- DNS resolving inside the Organization i.e. Zone specific.

The MX records fall in the Zone specific type which don't have a relative impact but other records do come under the Zone Transfer type and that is where the wild card has an impact. *As DNS is considered to be the backbone of the internet, risk can grow very quickly (or exponentially) depending on the wild card configuration in zone files.*

Search Engine Hacking – Traversing Deep for Information through Wild Cards

The Google search engine provides high-end working and information extraction functionality. With the advent of Google advanced search features, the searching process of information has elevated to a new standard. But the attackers are also using these features to find publicly available information which we term as reconnaissance. It has been observed that wild card plays a versatile role in search engine processes. Basically we are talking about the queries issued by an attacker or a normal person surfing for some information through search. Major search engines like Google, Yahoo, MSN etc provide advance keywords for effective searching. These keywords trigger the specific query by mapping with other keywords specified in one single query. As a result, a cumulative query will be sent to the search engine for finding requisite information. If we talk about Google, then Google Search Engine hacking is the term that is used. The GHDB (*Google Hacking Database*) is a collection of search strings derived with the keywords for finding information from the deeper parts of the internet. It works in a highly effective manner and is very rigorous. The wild cards again play a different role in search engine functionality.

For Example: If an attacker has to search for PHP pages in a domain and issues a request stated as:

```
Inurl:php site:domain.com or site:
   domain.com filetype:php
```

The search engine will display all the matches in the specific domain stated in the site parameter in the query. But this limits our search from finding information as it queries only the specific domain. The attacker can diversify this behavior by appending the `(*)` wild character in the site parameter:

```
Inurl:php site:*.domain.com or site:
   *.domain.com filetype:php
```

This not only searches for a domain but also for the entire sub domain that matches the wild card string. If a request is issued as:

```
Inurl:php? site:*.domain.com or site:
   *.domain.com filetype:php
```

After the 'or' the statement is the same as above. Is this correct?

The above stated query searches for the potential point. This means that the query will respond back with php? This all encapsulates entry related to php only. It makes the search engine to crawl more. Although certain features have been implemented as default but wild cards play an important role. The wild card usage has enhanced the search engine functionality thereby making it robust. But on the other hand it proves beneficial to attackers to try different combinations to extract the most information possible out of a single query.

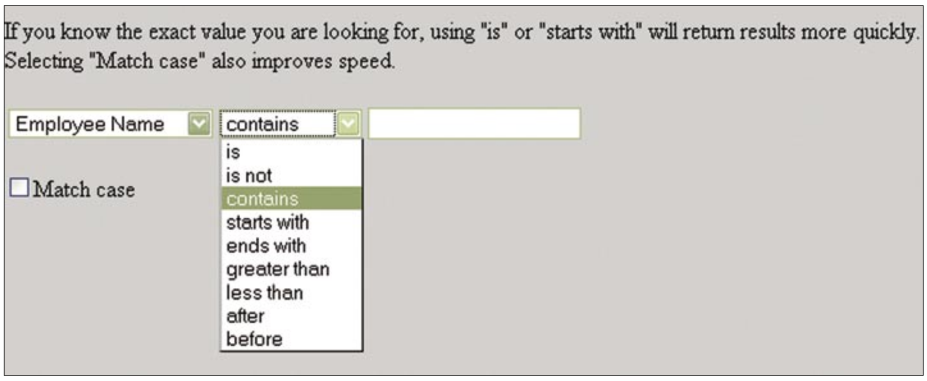

If you know the exact value you are looking for, using "is" or "starts with" will return results more quickly. Selecting "Match case" also improves speed.

Figure 2. *SQL Operators in Search Functionality*

Listing 1. *HTTP Verb Specification in Configuration File*

```
<security-constraint>
<web-resource-collection>
<web-resource-name>UserWR</web-resource-name>
<url-pattern>/listusers</url-pattern>
<url-pattern>/adduser</url-pattern>
<url-pattern>/addUserServlet</url-pattern>
<url-pattern>/deleteuser</url-pattern>
<url-pattern>/deleteUserServlet</url-pattern>
<url-pattern>/grantAccessServlet</url-pattern>
<url-pattern>/grantaccess</url-pattern>
<url-pattern>/removeAccessServlet</url-pattern>
<url-pattern>/removeaccess</url-pattern>
<url-pattern>/changeAccessServlet</url-pattern>
<url-pattern>/changeaccess</url-pattern>
<http-method>GET</http-method>
<http-method>POST</http-method>
</web-resource-collection>
<auth-constraint>
<role-name> * </role-name>
</auth-constraint>
<user-data-constraint>
<transport-guarantee>NONE</transport-guarantee>
</user-data-constraint>
</security-constraint>
```

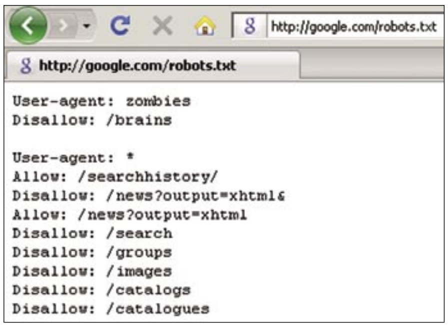
Wild Cards – Denial of Service in Database Querying

The wild cards are responsible for a number of different operations in databases. The queries that are used to automate the functioning of databases through the application layer depends a lot on wild characters. This is because SQL queries are inline. The SQL functionality covers the usage of wild characters at a higher level. A well crafted query with wild cards results in CPU consumption at a database level if a specific set of records are present. It's possible to exploit the built-in features of Microsoft SQL server which allows a user to design a query with wild cards. Let's look at the search functionality provided in an enterprise web application (see Figure 2).

One can notice the functionality provided to users for efficient research. Actually this problem has been found by researchers on the search page in a number of web applications running MSSQL server as the backend database server. The majority of the web applications provide an easy interface for the users to design a query. For Example: – a number of parameters are provided in the combo box right from the beginning. The user has to choose an option and provide the search string in the input search field. This is not only specific to the MSSQL server but other databases are also vulnerable. It depends on the parameter that is being used for the malicious query. The Like operator in MSSQL and MSACCESS, regexp operator in MYSQL and (~) operator in POSTGRESQL are vulnerable to this behavior. Using this operator with wild cards can impact the CPU usage and query time at a backend database level. The queries that impact the robustness of the application by hitting databases are mentioned bellow:

```
LIKE '%_[aaaaaaaaaaaaaaaaaaaaaaaaaaa
   aaaaaaaaaaaaaa[! -z]@$!_%'
LIKE '%_[~!@#$%^&*())(*&^%$$##@@@@
   @!%$^%$^%$&[! -z]@$!_%'
```

More details of this attack have been clearly stated in the paper [4]



```
User-agent: zombies
Disallow: /brains

User-agent: *
Allow: /searchhistory/
Disallow: /news?output=xhtml&
Allow: /news?output=xhtml
Disallow: /search
Disallow: /groups
Disallow: /images
Disallow: /catalogs
Disallow: /catalogues
```

Figure 3. *Robots File for Search Engines*

Again the wild characters vulnerability is used in a manner which leads to denial of service.

HTTP Verb Jacking – Wild Card Misconfiguration

The HTTP verb jacking allows an attacker to bypass the authentication and access control mechanisms. It has been noticed that the configuration file which is used to set the application access flow is not configured appropriately. The flaw persists in the specification of additional HTTP methods that are used to send requests to the server. It simply permits the unauthenticated access to resources if the file is not configured in an appropriate manner. The `web.xml` file is responsible for application level access. Let's understand how wild character presence impacts the state of the application. A sample target is selected through Google search engine (see Listing 1).

The above file shows the access control provided to the users. This file particularly possesses two problems from security perspective. The role name is provided with (*) wild character. There is no standard user who is configured like admin. The wild character presence shows that the access control is provided in a unanimous manner to all the users. It means there is no differentiation among the access rights. In addition to this, HTTP verbs are also not specified in an appropriate manner. The GET and POST request is specified for the request sent by the client. On the contrary, the other users can also use HEAD request to bypass access control on the above listed servlets. The problem can not be treated as normal because it marginalizes the robustness of an application. Everything needs to be explicitly defined in a well structured manner. But one can gauge the relative impact on the application flow when wild characters are specified in the misconfigured file. This in turn diversifies the attack surface.

Website Crawling – Usage of Wildcards in Robots.txt

The usage of wild cards in robots.txt file enhances the functionality and flexibility in matching the requisite strings for directories that are supposed to be crawled by the search engine. Let's have a look at the generic Google robots file. (see Figure 3)

The above presented snapshot describes the normal layout of robots.txt file. But inappropriate use of wild cards can dismantle the normal searching procedure and allow the search engine spiders to crawl for those destinations for which they not intended to be. Let's consider the wild card example in robots.txt file:

```
User-Agent: *
Allow: /public*/
Disallow: /*_print*.html$
Disallow: /*?sessionid
```

Now a days the major wild cards that are used in robots.txt are (*) and ($).The allowed parameter string is carrying a wild card which allows the search engine to crawl all directories starting with the public string. The presence of $ at the end of html will disallow all the requests by the search spider for files ending with html.

If the *robots.txt* file is not specified explicitly it can result in information leakage and path traversal to website directories through a search engine. Usually it is not considered as best practice but as a risky mechanism when designing the robots.txt file. Moreover, it requires a lot of testing after implementation prior to putting the website on the internet. As we know the robots file contains entries for allowing and disallowing pattern based mapping. The allow parameter enables the search spiders to crawl the pattern based objects and vice versa. Other problems that have also been noticed is the existence of duplication of records in a search engine lead by a mismanaged robots.txt file. Again, effective administration is required to combat this issue.

We have seen a number of security related problems in different domains due to wild card manipulation and its impact on numerous systems.

Conclusion

With the advent of the new techniques functionality has improved but at the same time the risk factor has also multiplied. This is because a transition has occurred from long procedures to a logical representation through pattern matching; using regular expressions and wild cards. The wrong implementation of these robust techniques impacts the functionality and behavior of running objects in a system. The risk becomes grave when another ingrained flaw in a component is fused with random logic i.e. wild cards usage etc. The inappropriate configuration is a relative part of it. This reflects the repercussions of the erroneous implementation of wild cards. Thus, in order to be secure, even smallest logic needs to be nurtured in the right manner.

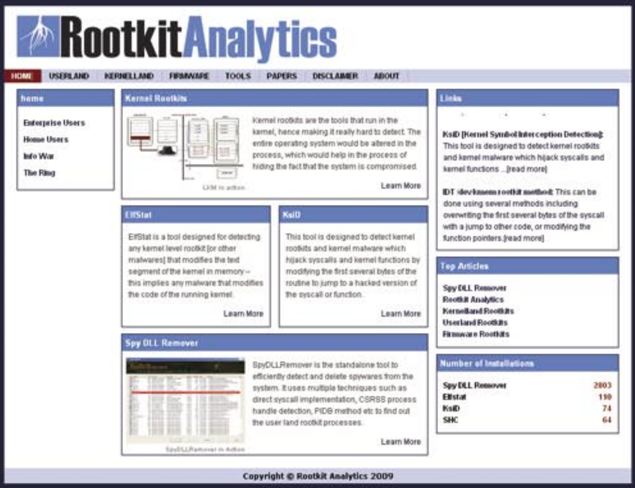
On the 'Net

- [1] *http://www.iab.org/documents/docs/2003-09-20-dns-wildcards.html*
- [2] *http://en.wikipedia.org/wiki/Zone_file*
- [3] *http://tools.ietf.org/html/rfc1034*
- [4] *http://www.portcullis.co.uk/uplds/wildcard_attacks.pdf*
- [5] *http://www.securiteam.com/unixfocus/5VP0K2KP6I.html*
- [6] *http://download.oracle.com/docs/cd/B28359_01/network.111/b28531/authorization.htm*

Aditya K Sood a.k.a 0kn0ck
Aditya K Sood is the founder of SecNiche Security. He is an independent security researcher having an experience of more than 6 years. He holds BE and MS in Cyber Law and Information Security. He is an active speaker at security conferences and already spoken at EuSecwest, XCON, Troopers, XKungfoo, OWASP, Club hack, CERT-IN etc. He has written journals for Hakin9, BCS, Usenix and Elsevier. His work has been quoted at eWeek, SCMagazine, ZDNet, internet news etc. He has given a number of advisories to fore front companies. On professional front he works for KPMG as a penetration tester.
Website: http://www.secniche.org
l Blog: http://zeroknock.blogspot.com

RootkitAnalytics

www.RootkitAnalytics.com


RootkitAnalytics

HOME | USERLAND | KERNELLAND | FIRMWARE | TOOLS | PAPERS | DISCLAIMER | ABOUT

home

Enterprise Users
Home Users
Info War
The Ring

Kernel Rootkits

Kernel rootkits are the tools that run in the kernel, hence making it really hard to detect. The entire operating system would be altered in the process, which would help in the process of hiding the fact that the system is compromised.

LKM in action

Learn More

ElfStat

ElfStat is a tool designed for detecting any kernel level rootkit [or other malwares] that modifies the text segment of the kernel in memory -- this implies any malware that modifies the code of the running kernel.

Learn More

KsiD

This tool is designed to detect kernel rootkits and kernel malware which hijack syscalls and kernel functions by modifying the first several bytes of the routine to jump to a hacked version of the syscall or function.

Learn More

Spy DLL Remover

SpyDLLRemover is the standalone tool to efficiently detect and delete spywares from the system. It uses multiple techniques such as direct syscall implementation, CSRSS process handle detection, PIDB method etc to find out the user land rootkit processes.

SpyDLLRemover in Action

Learn More

Links

KsiD [Kernel Symbol Interception Detection]: This tool is designed to detect kernel rootkits and kernel malware which hijack syscalls and kernel functions ...[read more]

IDT /dev/kmem rootkit method: This can be done using several methods including overwriting the first several bytes of the syscall with a jump to other code, or modifying the function pointers.[read more]

Top Articles

Spy DLL Remover
Rootkit Analytics
Kernelland Rootkits
Userland Rootkits
Firmware Rootkits

Number of Installations

Spy DLL Remover	2803
Elfstat	110
KsiD	74
SHC	64

Copyright © Rootkit Analytics 2009

Rootkit Analytics - the science of rootkit analysis, is a web-portal sculptured to enhance research, analysis and development of rootkit defense mechanisms. There are many unsolved problems in this world, one of which has been rootkit analysis. This is due to the fact that the ongoing war between good and the bad is never ending. With that being the case, Rootkit Analytics is aiming to provide solutions and services in analysis and mitigation of rootkits.

contact.fingers@gmail.com