

IIS Workflow Described

Rainer Gerhards

rgerhards@adiscon.com

Adiscon GmbH, Mozartstrasse 21, 97950 Grossrinderfeld, Germany

Last modified 20/03/2003 12:21

Abstract

This paper describes the IIS workflow (aka “order of operations”) as far as the author **thinks** it is. I have tried hard to make the information as complete and accurate as possible, but obviously it might be wrong as I haven’t coded IIS. All information in this paper is taken from official Microsoft documentation as well as working with IIS in lab.

This paper will become part of a larger IIS-logging focussed paper. In fact, I excerpted it from a later release, because I see increasing interest in IIS workflow caused by some new malware on the web. So this paper is not fully proof-read and there will most probably be some typos, grammatical errors as well as inconsistencies. Especially if you spot an inconsistency, I would appreciate if you let me know at rgerhards@adiscon.com because I obviously would like to have it as easily understandable as possible.

The above mentioned IIS logging paper can be found at <http://www.monitorware.com/Common/en/SecurityReference/monitoring-iis-logs.pdf>

Additional information and papers may also be available at <http://www.monitorware.com/Common/en/SecurityReference/>.

Please note that this document is correct for IIS up to version 5.1. Microsoft has announced considerable change for IIS 6.0, and the information in this document might not be valid for that version.

IIS order of Operations

Conceptually, IIS has three processing stages:

1. Decoding the target (web page) of the request
2. Serving the request
3. Finishing the request, which **includes** logging

Before we dig into the details, be sure to fully understand the implications of that overall order of processing: **no** data is logged by IIS before the request is finished. The reason for this is that IIS log files contain data like the total number bytes send or the processing time required. This data is

only available when the request is finished. This also means that when an attacker either crashes IIS or exploits a vulnerability that does not allow it to reach step 3, no log data is written. As a side note, the same is true for Apache.

The problem of missing log data from web servers failing in stages one or two can only be addressed by writing two log lines – one at the begin of the request and one at the end. This is currently not supported by IIS (nor other web servers I know of). Adiscon is developing a tool that allows to do this. Please email info@adiscon.com if you are interested in this and would like to receive a copy as soon as it is available.

Now let us dig into the details of the IIS order of operations. A word of warning first: We have compiled this information from what is published in the Microsoft IIS SDK as well as our own testing. We made any attempt to do do it correctly. However, there is a slight chance that we are failing in one regard or another, so please be careful when basing your work on the information contained in here. Depending on what you try to do, it can be a good idea to re-confirm the information in question with somebody at Microsoft.

First of all, it is important to remember that http request come in via TCP. TCP is a stream-oriented protocol, not a packet-oriented one. This means that any TCP application (like IIS) needs to expect that multiple reads of the TCP stream can be necessary to read a piece of information – even if it is a small one. A practical sample: the URL of the requested page is contained in the very first bytes of the HTTP header. So typically, this information is readily available when IIS receives the first TCP packet and begins processing of it. **However**, it is possible that someone manually connects to IIS via telnet to port 80. As he then types the HTTP header on the keyboard, **each** character will be delivered in its own packet. As such, the initial packet will just hold **one** character and **not** the complete URL that is typically in there. We need to keep this in mind when talking about the IIS order of operations. Some of the stages described here will typically only be involved once, but there might be situations where multiple executions of these stages code base might be necessary. This is more an implementors issue, but it can be a challenge to log analysis and securing IIS as vulnerabilites, too might be caused by implementors not appropriately handling such situations (this especially applies to ISAPI filters, which require a deep understanding of what IIS does).

Stage 1: Decoding the Request

This initial phase is used to read the HTTP headers and identify the page (or script) to be called by the web server. This phase also involves authentication for non-anonymous access.

IIS reads the TCP stream until sufficiently information has been gathered. “Sufficiently” means at least the actual request type (line one of the HTTP header) as well as the host headers and such. I did not find a clear indication of what “sufficient” is. In my point of view, I think it is the complete HTTP header, but I can not prove this to be 100% exact.

Please note that when “NT Challenge / Response” authentication is used. IIS will even initiate the challenge / response process with the client. This involves sending the challenge and receiving the response, two network packets, as part of the initial phase.

As soon as IIS has read enough of the request, it begins to process the headers. At this point, the unprocessed headers are already in a memory buffer.

Though not documented by Microsoft, I think the first thing done is to extract the HTTP host header, which is then used to obtain configuration information from IIS' metabase. The host header is vitally important, as all processing and analysis – even the further decoding – is depending on the virtual host being requested. The information from the host header will be used to obtain configuration information during all stages of IIS processing, not just decoding. If there is no (valid) host header, then the default web site is used. If there is no default web site and also no web site configured specifically for those requests without a (valid) host header, the request is simply discarded at this stage. Please note that this also means no log entry will be written as this is done in stage 3.

Then, the final URL is decoded. Depending on the ISAPI filters installed, this can be a lengthy process. In its most basic form, the URL decoder looks into the supplied HTTP host header first and obtains the path mapping from the IIS metabase. Please note that not only the configured home directory plays a role in this calculation but also configured virtual directories, if any. Lastly, ISAPI filters have the ability to change the URL decoding in any way they like. As multiple ISAPI filters are loaded, multiple interim decodes may happen. In any case, at the end of this process the URL is fully decoded and pointing to an actual file system path of a file to execute (or deliver, in case of a static page).

Then, authentication is performed. The result of this operation is the user context into which the request should be served. That context is later on used for all permission checks. Please note that in IIS, other than in Apache, every request is executed in the security context of a specific windows user account. This also applies to anonymous requests. In MMC, you can configure the account used for anonymous requests. By default, this is called IUSR_<Machinename>. There may be other default users, depending on the exact technology and version used (for example, ASP.NET defines another anonymous user account).

Once the user is known, IIS checks the NTFS ACLs (if any) of the previously decoded actual file to be executed. If the authenticated user does not have proper privileges, IIS denies access to it. Please note that it looks like IIS does this check by itself. It is **not** relying on the operating system to check if the authenticated user has proper permissions. A possible way to do so would have been to impersonate as the authenticated user and try to access the file. However, IIS at this stage is not yet impersonated as the authenticated user. As such, this mode to check the file permissions seems not to be doable. On the bottom line this means that if there is a bug in IIS' permission checking, the operating system itself is out of luck and can not detect that. Former vulnerabilities (especially the Unicode Path Mapping vulnerability) prove this observation and also show the damage potential this way of processing has.

As written above, authentication can be quite complex, especially if Microsoft proprietary methods are used. If the user can not properly be authenticated, a "Request denied" page is served back to the requestor. In this case, the request is simply not served, which means stage 2 below is not executed. However, stage three **will** be used in this case and as such logging of invalidly authenticated requests will happen.

Please note the fine difference: if the authentication fails, IIS continues to work and just does not execute the request. If, however, a vulnerability is exploited during this stage, IIS will probably not continue normally and the request will most probably never be logged.

Once this is done, IIS immediately seems to begin reading further incoming data (for example in a post stream). It looks like this is asynchronous to actual request execution (but I have not verified this).

Stage 2: Serving the Request

Serving the request is easy once the incoming parameters are set.

IIS is a multithreaded application. It has a pool of so-called worker threads. One of these threads is assigned the incoming web request for processing. If all worker threads are currently serving incoming (unfinished) requests, IIS typically just creates a new worker thread to serve the new request. In other cases, IIS does not create a new worker thread but suspends processing of the incoming request until a new worker thread is available. Which method IIS chooses is depending by the IIS configuration, machine utilization and machine resources (and maybe some other factors I don't know about). I am also not sure if the worker thread is assigned beginning in stage 2 or right at the start of the processing sequence, at the start of stage 1 above. In any case, the actual processing will be carried out in the worker thread, and this is important for our description.

Before actually serving the request, IIS checks what kind of processing it must do. There are three potential cases:

- Static files
- Script files
- Executable files

Static files are served by IIS itself. No other software is involved in serving static files. However, ISAPI filters are still involved while static files are being served, so there is a potential for failure here as well as in core IIS. I deem the potential in core IIS to be very minimalistic. Please note that no external process is started when serving static files.

Script and executable files are processed in more or less the same way, thus I describe them together. An executable file is a file that is executable by itself, like an (CGI) exe file. A script file needs the help of a script processor. Examples of the later are ASP or PHP scripts – and also perl.

A new process needs to be created for all files where the script processor is not available as an in-memory ISAPI DLL (ASP and ASP.NET or such DLLs). Typical examples for non.resident script processors are PHP and perl. In their case, an external process needs to be created. From IIS' point of view, it does not really make a difference if the new process is created for a script processor or for a “real” executable. A script processor is just an executable that needs to be passed in some parameters in a specific way.

To create a new process, IIS first creates a new process space. This is still done in the security context of IIS itself, most often the SYSTEM account. Then, IIS loads the image file to be executed into that still-empty process stage. Again, this is done by IIS itself. Consequently, file system audit events 560 of source “Security” in the Windows Security Event Log do show the

account IIS is running under as the process initiator. This can be misleading when doing analysis of the event logs. When the image is loaded **then** IIS impersonates into the context of the authenticated user and changes the new processes access token to be that of the authenticated user. If you are analysing this in the Windows Security Event log, there should be a event id 600, source "Security" that holds this information. IIS then runs the newly created process.

The new process communicates via defined interfaces (e.g. CGI) with IIS and the web client. Depending on the actual interface used, IIS asynchronously reads and writes web client data.

There is at least one difference between scripts and "plain" executables: for scripts, there is a timeout in the IIS configuration. If scripts don't finish within a configured amount of time, they are aborted (at least ASP scripts, not sure for all potential scripting processors). "Real" executables, do **not** have any runtime restriction. In lab, we have run ISAPI DLLs (a form of executable) for more than one day in a single IIS thread. IIS did neither abort that executable nor log any error or warning message.

This behaviour can be used in an attack: An attacker exploits a vulnerability that allows him to inject code into one of the worker threads. This code will do malicious things **and** run in a tight loop (much as SQL Slammer [5]did). As such, it will never finish, thus the worker thread will also never finish. What does this mean? Well, we are stuck at stage two and will never reach stage 3, so no logging will happen.

When the executable (or script processor) is done with the request, it terminates and control returns to the calling worker thread. Please note the process termination will be logged in the correct user context in Windows Security Event log. The event id logged is 593 from source "Security".

Stage 3: Finishing the Request

Once the request has been served, IIS does some cleanup work. Part of that cleanup work is logging. **Now** IIS has available all information that it potentially needs to write to the log file. Things like the return status of the process executed, the number of bytes sent or the total execution time. It is for this reason that IIS does **not** log data at any further stage.

Please note that ISAPI filters can modify IIS' logging behaviour. If requested by a filter, IIS will call the filter right before the log entry is written. The filter can then decide to modify or even drop the log data.

When the filter returns, IIS finally writes the line to the log entry. This also implies that sequence of events in the log file is based on the time they are **finished**, not initiated. An example: if request A starts at 9:15:01 and ends at 9:15:05 and request B starts at 9:15:02 and ends at 9:15:03, the sequence in the log file will be B first, then A. The reason for this is that A finished after B and the log file sequence is based on the time the request finished. This fact is worth noting, as the actual processing sequence can not accurately be guessed from web server logs when doing forensic analysis. This most probably applies to all web server logs, not just IIS logs.

Copyright

This document is copyrighted © 2003 by Adiscon GmbH and Rainer Gerhards. Anybody is free to distribute it without paying a fee as long as it is distributed unaltered and there is only a reasonable fee charged for it (e.g. a copying fee for a printout handed out). Please note that “unaltered” means as either the PDF file you (hopefully) acquired or a printout of the same on paper. Any other use requires previous written authorization by Adiscon GmbH and Rainer Gerhards.

If you place the document on a web site or otherwise distribute it to a broader audience, I would appreciate if you let me know. This serves two needs: Number one is I am able to notify you when there is an update available (that is no promise!) and number two is I am a creature of curiosity and simply interested in where the paper pops up.

Credits

Many thanks to Tina Bird of www.loganalysis.org, which had the idea to this paper and provided numerous feedback and tough questions. Without that, it wouldn't exist at all.

Authors Address

Rainer Gerhards
rgerhards@adiscon.com

Adiscon GmbH
Mozartstrasse 21
97950 Grossrinderfeld
Germany

Disclaimer

The information within this paper may change without notice. Use of this information constitutes acceptance for use in an AS IS condition. There are NO warranties with regard to this information. In no event shall the authors be liable for any damages whatsoever arising out of or in connection with the use or spread of this information. Any use of this information is at the user's own risk.