



TrustLeap[®]

Global-WAN[®] (G-WAN v1.0)



User's manual

Copyright notice

All mentioned trademarks and brands are the property of their respective owners.



Disclaimer and Legal Information

NO LICENSE, EXPRESS OR IMPLIED, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN TRUSTLEAP'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, TRUSTLEAP ASSUMES NO LIABILITY WHATSOEVER, AND TRUSTLEAP DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF TRUSTLEAP PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY TRUSTLEAP, TRUSTLEAP PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE TRUSTLEAP PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

TrustLeap may make changes to specifications and product descriptions at any time, without notice. TrustLeap shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes. The information here is subject to change without notice. The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications.



Summary

This document consists in three Chapters:

- I. web server,
- II. dynamic contents,
- III. advanced features.

It is recommended to read the user's manual in this order: some options are *transparently* handled and may appear to be missing to anyone looking for a setup program or a configuration file (G-WAN version 1.0 uses none).

The same is true for servlets: useful features are sometimes invoked by *not* doing something rather than by calling a dedicated function.

G-WAN's main design rule is "simplicity rules". Computers hate complexity almost as much as humans. G-WAN tried to make them all a favour in this matter.

G-WAN offers a very stable, safe and fast web server designed to scale *as well as possible* with the many small GET/POST HTTP requests typically used by Web 2.0 (but studies show that 90% of all static contents are smaller than 100KB).

G-WAN is *provably* safer than other web servers for at least two reasons:

- it contains much less lines of code (which also means far less bugs);
- it does not use libraries and does not copy buffers (no more stack exploits).

These unusual characteristics also make G-WAN unusually more efficient.

While G-WAN is fast with large files, it can really shine with small files only: web servers don't *receive* or *send* data: operating systems do it. If you serve a 1MB file, the CPU time used by G-WAN is negligible compared to the time used by the system to send data to the network. Small files (and HTTP keep-alives) let G-WAN better make the difference because a larger part of the work is done by G-WAN.

Web servers only *parse* requests and *build* replies. And G-WAN is doing it properly. So, if you mostly serve KB-files (instead of MB-files) then G-WAN will still work seamlessly under loads that will cause others to stop responding.

But G-WAN can also *process* data (rather than delegating this task).

G-WAN's platform is designed for *highly-scalable* and *low-latency* web applications. The kind that you simply could not make before on a single machine.



Foreword

This is the first public release of a completely new HTTP server.

While we did our best to check it against all the possible attacks we could think about, the only reliable test in this matter is a public exposure.

So, please, download G-WAN, install it and attack it *on your PC* by all the possible ways you can. Creativity is welcome as long as you don't harm people.

If you find a security problem then send us all the details at <http://trustleap.com> so we can take action without delay.



I. The web server

Installation and startup

To install G-WAN, download it from gwan.com, copy the zip archive in a folder of your hard-disk (like C:\G-WAN), decompress it and run the gwan.exe program.

Once you have done that, if you want to benefit from the best possible performances you will have to reboot your PC in order to let gwan tune the Windows TCP/IP stack.

Decompressing the zip archive will create the following sub-directories:

C:\G-WAN\www	where you will copy HTML files and image files
C:\G-WAN\csp	where you will copy C servlets (servlets are explained in Chapter II)

To test the server, open a web browser and enter the following address:

<http://127.0.0.1>

If the welcome page is not displayed then ask someone to assist you with the network issues (they largely go out of the scope of this manual).

By default, the server will listen to port 80, on all network interfaces.

To make it listen to specific interfaces and port numbers, use the following command-line options:

usage: gwan [-listen:<addr|'*'>[:port]]...

where <addr> can be '*' to use all the interfaces,
[port] is a (1-65535) number (80 by default)

examples: gwan (without parameters)
gwan -listen:*.443
gwan -listen:192.168.7.8:443

Listeners will not use CPU resources if no incoming connections are hitting the server.

Log files

G-WAN can use traditional (Apache-like) log files. To activate this feature, just create a sub-folder called C:\G-WAN\logs. Log files will not be generated if the folder does not exist (or exists under another name).



G-WAN's performances are slightly lower when log files are enabled, so, if you are doing benchmarks or if you are aiming for the highest possible scalability then you should think about using a reverse proxy dedicated to logging rather than wasting CPU on your origin servers.

Supported HTTP features

Protocols	HTTP/0.9, HTTP/1.0, HTTP/1.1
Methods	GET, HEAD, POST (application/x-www-form-urlencoded), PUT, DELETE, OPTIONS
Encodings	"entity" (but all encodings are parsed for <i>filters</i> to support them)
Conditions	If-[Un]Modified-Since

This is for G-WAN version 1.0: future versions may gradually add other HTTP features.

Supported MIME types

```
{ 0, "", "application/x-msdownload" }, // fall-back value
{ 0, "mp3", "audio/mpeg" },
{ 0, "wav", "audio/wav" },
{ 0, "avi", "video/x-msvideo" },
{ 0, "mov", "video/quicktime" },
{ 0, "flv", "video/x-flv" },
{ 0, "mng", "video/x-mng" },
{ 0, "mpeg", "video/mpeg" },
{ 0, "mpg", "video/mpeg" },
{ 0, "asx", "video/x-ms-asf" },
{ 0, "wmv", "video/x-ms-wmv" },
{ 0, "bin", "application/octet-stream" },
{ 0, "exe", "application/octet-stream" },
{ 0, "dll", "application/octet-stream" },
{ 0, "swf", "application/x-shockwave-flash" },
{ 0, "der", "application/x-x509-ca-cert" },
{ 0, "pem", "application/x-x509-ca-cert" },
{ 0, "crt", "application/x-x509-ca-cert" },
{ 0, "ps", "application/postscript" },
{ 0, "eps", "application/postscript" },
{ 0, "ai", "application/postscript" },
{ 0, "js", "application/x-javascript" },
{ 0, "json", "application/json" },
{ 0, "atom", "application/atom+xml" },
{ 0, "rss", "application/rss+xml" },
{ 0, "rtf", "text/richtext" },
{ 0, "txt", "text/plain" },
```



```
{ 0, "zip",      "application/octet-stream"    },
{ 0, "pdf",     "application/pdf"      },
{ 0, "tif",     "image/tiff"          },
{ 0, "bmp",     "image/x-ms-bmp"      },
{ 0, "svg",     "image/svg+xml"       },
{ 0, "css",     "text/css"            },
{ 0, "jpeg",   "image/jpeg"          },
{ 0, "jpg",    "image/jpeg"          },
{ 0, "png",    "image/png"           },
{ 0, "gif",    "image/gif"           },
{ 0, "html",   "text/html"           },
{ 0, "htm",    "text/html"           }
```

As this list is hard-coded you cannot add MIME types in G-WAN version 1.0.

If you need a way to complete this list (from C servlets for example), just let us know. To keep things simple, we try to avoid configuration files but if there's a demonstrated need for it then we will implement it.

Default style sheet and HTTP Errors

To personalize the HTTP default style sheet, you have to make your CSS style available under C:\GWAN\www\imgs\style.css.

While G-WAN is supporting *all* the HTTP error codes (that's useful for C servlets), only a subset is relevant for the server (like Not found, Internal error, etc.).

To personalize the HTTP error style, you have to create a CSS style sheet and make it available under C:\GWAN\www\imgs\errors.css.



II. Dynamic contents

Web Servers need *scripting* capabilities for convenience and *hard-coded filters* for performances. G-WAN does both in C -with compiled code performances.

How many languages do you need to learn if one works better than others? C made Unix, Windows, games, PDF viewers, web browsers. C servlets will be limited by your sole imagination. C survived 40 years for a reason: it fits the task.

Some will say that C lacks garbage collectors and crash-proof error-recovery. G-WAN's memory pools and 'graceful' crash handling should calm their fears.

Assuming G-WAN is installed and running, if you look at the files located in the `./csp` directory, you will see small C source code files (that we call "servlets").

C servlets are executed by G-WAN when a client requests the corresponding URL: <http://127.0.0.1/csp?bench>

The server will return the `bench.c`'s "reply" buffer to the client that sent this query.

Your first servlet: "301 moved permanently"

Redirecting users is useful after you moved or deleted the previous URI on your server. We want to send the following to clients asking for the old URI:

All the information necessary for a redirect is in the headers. The body of the response is typically empty, but we will create one in order to see how to proceed:

```
int main()
{
    static char szURI[]="new.html"; // new location

    // create a dynamic buffer and get a pointer on the server response buffer
    xbuf_ctx reply; get_reply(argv, &reply);

    xbuf_xcat(&reply, "HTTP/1.1 301 Moved Permanently\r\n"
               "Content-type: text/html\r\n"
               "Location: new.html\r\n\r\n"
               "<html><head><title>Redirect</title></head>"
               "<body>Click <a href=\"%s\">here</a>.</body></html>",
               szURI);

    // (they have changed when more memory is allocated during formatting)
    set_reply(argv, &reply); return(301); // return an HTTP code (301:'Moved')
}
```




The function `xbuf_xcat()` works like `sprintf()` and lets you write the reply that the server will send to the client (without worrying about the length of the buffer).

Your “reply” buffer can contain HTTP headers only, or just HTML code and no headers, or both headers and HTML. When HTTP headers are missing, the server creates headers to match your `main()`'s return code.

All the standard HTTP status codes are supported but if you use your own custom codes (in the 600+ range) then the server can't imagine their purpose so you will have to explicitly define headers *and* HTML (if you target human clients).

The following example (without headers) is equivalent to the previous example (which explicitly defined response headers):

```
int main()
{
    static char szURI[]="new.html"; // new location
    xbuf_ctx reply; get_reply(argv, &reply);

    xbuf_xcat(&reply, "<html><head><title>Redirect</title></head>"
              "<body>Click <a href=\"%s\">here</a>.</body></html>",
              szURI);

    set_reply(argv, &reply); return(301); // return an HTTP code (301:'Moved')
}
```

A servlet can use this auto-completion feature to reduce the code to its simplest expression (for example, to filter connections per IP address, CIDR, or country):

```
int main() // status code 401 means 'Unauthorized'
{
    ...           // do whatever you need to filter connections
    return 401; // the server will use this code to build headers and an HTML reply
}
```

At the moment, servlets cannot “insert” headers to the server reply (Request Handlers can do it): either your servlets will define all the headers or you will expect the server to do it all for you. Environment variables (like an up-to-date HTTP date stamp) are available to make it easier to build HTTP headers.

Note: To send something else than HTML (like a PNG or an XML document), you **MUST** explicitly define HTTP headers (servlet examples are provided).

Other dynamic buffer `xbuf_xxx` routines will help you in the task of building a reply. Let's quickly review them to understand their purpose.



xbuffer dynamic buffers

Dynamic buffers, like memory pools, are an efficient way to reduce the burden of memory management for high-performance programs. They are also convenient: servlets can just fill dynamic buffers without having to care about size, alignment, allocation lifetime, locks or heap fragmentation.

They are also safer: you can't overflow dynamic buffers (unless you are using all the memory available on a machine).

Each C servlet has an xbuffer called "reply" aimed at sending information to clients.

But it may also be useful to create additional dynamic buffers in your servlets (to load an HTML template file, or to get the reply of a query sent to a web server).

You are expected to call `xbuf_free` to release any memory you have used (but don't free the "reply" buffer!).

<code>xbuf_reset()</code>	(re)initialize a dynamic buffer (without freeing memory)
<code>xbuf_frfile()</code>	load a file, and store its contents in a dynamic buffer
<code>xbuf_tofile()</code>	save the dynamic buffer in a file
<code>xbuf_frurl()</code>	make an HTTP request, and store the result in a dynamic buffer
<code>xbuf_cat()</code>	like <code>strcat()</code> , but in a dynamic buffer rather than a string
<code>xbuf_ncat()</code>	like <code>strncat()</code> , but can also copy binary data in the specified buffer
<code>xbuf_xcat()</code>	formatted <code>strcat()</code> (<i>a la printf</i>) in the specified dynamic buffer
<code>xbuf_insert()</code>	insert bytes at a given position in the buffer
<code>xbuf_delete()</code>	delete bytes at a given position in the buffer
<code>xbuf_getln()</code>	get an LF-terminated text line from a buffer
<code>xbuf_findstr()</code>	find a given string into the buffer
<code>xbuf_repl()</code>	replace a string by another string in a buffer
<code>xbuf_free()</code>	release the memory previously allocated for a dynamic buffer
<code>xbuf_replfrto()</code>	like the call above, but from/to given pointers in the buffer

All the servlet samples (/csp folder) demonstrate the syntax of those functions.

But sending information is only half of the job: often, you will also need to get information sent by the client (via GET or POST HTTP requests).

Getting GET/POST parameters

G-WAN transparently process both in the very same way to let you access any passed parameter with the same code (via the `get_arg()` call).

Please refer to the `csp/contact.c` and `csp/loan.c` servlets for real-life examples. You can invoke those samples as follows:

<http://127.0.0.1/csp?contact> and <http://127.0.0.1/csp?loan>



Getting server “environment” variables

Traditional 'environment' variables are available to servlets. Additions, like the current Http date have been added: as the work is already done by the server, there is no need for servlets to do it again.

Please refer to servlet samples like `csp/contact.c` for a list of all environment variables and how to access them (via the `get_env()` call).

Additional functions

The calls below are provided for your convenience:

<code>cycles()</code>	get the CPU clock cycle counter's value (32-bit)
<code>cycles64()</code>	get the CPU clock cycle counter's value (64-bit)
<code>get_arg()</code>	get GET/POST application/x-www-form-urlencoded parameters
<code>get_env()</code>	get G-WAN's “environment” variables
<code>url_encode()</code>	properly encode an URL so you can use it
<code>escape_html()</code>	properly encode a buffer so you can use it in HTML
<code>html2txt()</code>	remove all HTML tags from a buffer
<code>s_time()</code>	equivalent to <code>time(0)</code> ; but faster
<code>s_asctime()</code>	equivalent to <code>asctime()</code> ; but faster (and thread-safe)
<code>time2rfc()</code>	format an Http date string from a given <code>time_t</code> value
<code>rfc2time()</code>	return a <code>time_t</code> value from an Http date string

Putting it all together

G-WAN's samples are available under the `/csp` sub-folder. The most advanced sample is `loan.c`, which uses AJAX to process a form without reloading the page:

Loan Calculator

Please fill the fields and press the 'Calculate' button ('*' fields are mandatory):

Your Full Name	<input type="text" value="Philippe Martin"/>
* Amount	<input type="text" value="1000000"/>
* Rate	<input type="text" value="3.5"/> %
* Years	<input type="text" value="1"/>
<input type="button" value="Calculate"/>	



When users press the 'Calculate' button, the loan is displayed in the same page:

Loan Calculator

Please fill the fields and press the 'Calculate' button (*' fields are mandatory):

Your Full Name

* Amount

* Rate %

* Years

Dear Philippe Martin, your loan goes as follows:

LOAN	DETAILS
Amount	1,000,000.00
Rate	3.50%
Term	1 year(s)
Cost	19,059.56 (1.91%)

YEAR 1				
MONTH	PAYMENT	INTEREST	PRINCIPLE	BALANCE
January	84,921.63	2,916.67	82,004.96	917,995.04
February	84,921.63	2,677.49	82,244.14	835,750.89
March	84,921.63	2,437.61	82,484.02	753,266.87
April	84,921.63	2,197.03	82,724.60	670,542.27
May	84,921.63	1,955.75	82,965.88	587,576.39
June	84,921.63	1,713.76	83,207.87	504,368.52
July	84,921.63	1,471.07	83,450.55	420,917.97
August	84,921.63	1,227.68	83,693.95	337,224.01
September	84,921.63	983.57	83,938.06	253,285.95
October	84,921.63	738.75	84,182.88	169,103.07
November	84,921.63	493.22	84,428.41	84,674.66
December	84,674.66	246.97	84,427.69	0.00

This page was generated in 346,856 cpu clock cycles.
(on a 3GHz CPU 1 ms = 3,000,000 cycles)

As this C servlet is resolving a real-life problem it can be used as a benchmark for



dynamic contents (Webspec is so large that it a lot of requires time to implement).

It would be great to see how the *very same** implementation performs in, say, Perl, PHP, Lua, TCL, Python, Java or ASP.Net.

(*): same means using the same style sheet, number decoration, formatting, inputs, outputs and general program structure (no pre-calculated loans please).

Another advantage of using AJAX here is the fact that it saves you from using `mod_rewrite` to have "normal" search-friendly URLs while taking advantage of the server-side functionality of servlets.

Execution errors, crashes and debugging

G-WAN will signal syntax errors, undefined symbols, etc. before the code executes. G-WAN will also "gracefully" handle C servlet crashes and report where the fault happened (instead of crashing the server). If you let G-WAN run this code:

1. `void crash() { *((int*)(0))=0xBADC0DE; } // write access violation`
2. `int main () { crash(); return(200); }`

G-WAN will tell you *what line in your source code file* did it wrong:

```
Exception      : c0000005 Write Access Violation
Address       : 06d3b413
Access Address : 00000000
```

```
Registers : EAX=0badc0de CS=001b EIP=06d3b413 EFLGS=00010246
            EBX=00000000 SS=0023 ESP=0166df34 EBP=0166df3c
            ECX=00000000 DS=0023 ESI=00000104 FS=003b
            EDX=0166fc58 ES=0023 EDI=0166f47c CS=001b
```

```
Call chain  :(line) PgrmCntr(EIP) RetAddress FramePtr(EBP) StackPtr(ESP)
crash():    1    06d3b413  06d3b4a6    0166df3c    0166df34
main():     2    06d3b4a6  0042d1ea    0166df64    0166df34
```

```
Servlet: csp/crash.c
Query : /csp?crash
Client : 127.0.0.1
```

Until you fix the code, G-WAN tells clients that the service is unavailable (503). Instead of documenting potentially dangerous holes, bugs will just 'not exist'.

Web Applications Security

Cross-site scripting, injection attacks or request forgery are made easy and having success for simple reasons.



They can easily be pinpointed:

- the surface of vulnerability is expanding with new browser features;
- web developers already have a job and just can't cope with these issues;
- fixing the whole stuff would severely harm the advertising business.

An efficient solution, however, can be implemented without changing anything to the current happy mess.

It just requires proper cryptographic-grade Session tokens, ID tokens and HMACs (transparently implemented *by the server* to make sure that they will be used).

Cryptography is typically weakened or avoided to avoid harming performance or scalability. This is mainly due to the fact that people reuse generic code instead of writing on-purpose code.

As G-WAN illustrated it, there is room for improvement in this matter.

Any help will be welcome and appreciated. Money buys time -and time is needed to do good things.



Feedback

Any useful suggestion is welcome, but as our time is limited try to follow the guidelines below:

- use a relevant subject in your email so we know what you want,
- please go straight to the point and give a **real-life** example,
- be kind: it's version 1.0 so there is *obvious* room for enhancements.

If many software vendors do not let you contact them (or do it in a way that defeats the purpose), there is a reason: this is a very time-consuming process.

The only way to keep this service available is to respect its constraints.