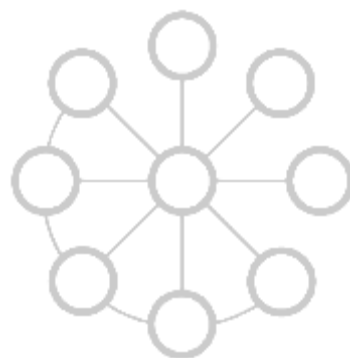




# TrustLeap<sup>®</sup>

**Global-WAN<sup>®</sup>**  
(G-WAN v1.0.5)  
*for Linux & Windows*



## *User's manual*

*"Simplicity is the ultimate sophistication."*  
(Leonardo da Vinci, 1452-1519)

*"Complexity is the enemy of security."*  
(Bruce Schneier, 1963-)



## Summary

This document consists in three Chapters:

- I. web server,
- II. dynamic contents,
- III. extending the joy.

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 spirit is "simplicity rules". Computers hate complexity almost as much as humans. G-WAN tries to make them all a favor 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 the Web (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 only really shine with small files: 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 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 cause others to stop responding.

But G-WAN can also *process* data (rather than just 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 -and a single developer.



## Foreword

G-WAN is a new HTTP server published for the first time in July 1st 2009.

### **PROVEN SECURITY:**

Since then, G-WAN has been continuously attacked (without success) by thousands of people from all around the Planet (but mostly from the USA and U.K.). Both script kiddies and well-funded reputed experts (sometimes hired by the competition) did their very best at trying to crash G-WAN -*without success*.

Given the load (at times, more than 80% of the traffic was only made of attacks targeting G-WAN's sample servlets, API calls, directory paths and timeouts), it is not irrelevant to say that G-WAN is safer than its unfortunate predecessors -as none of them survived their first public exposure.

Less code also means less bugs: studies have shown that software averages 1-16 bugs per 1,000 lines of code (<http://www.minix3.org/doc/reliable-os.pdf>).

If you find a security problem then send us the details so we can take action without delay. *All valuable contributions will receive full credits on [trustleap.ch](http://trustleap.ch).*

### **DENIAL OF SERVICE (DoS) PROTECTION:**

At the beginning G-WAN was as fast as possible. When it was published it was attacked by every possible means. If your HTTP server is fast and safe, poisoned URLs just cost bandwidth. But *timeout attacks* are targeting the system rather than server applications. And they can make your operating system crash.

Timeout attacks are simple: a connection is established normally and, suddenly, the client stops sending/receiving data (or does it much smaller than before). They are also *invisible* because at best they appear as legit traffic in server logs.

*The day before Slashdot's article*, G-WAN received no less than 82,568 timeout attacks (yes, they are logged):

Accept: 80,078    Read: 321    Slow: 37    Send: 2,132

Without G-WAN's DoS protection, those 82,566 would have piled to millions and saturated the system memory (each connection is using system resources).

But G-WAN/Windows' DoS protection defeats benchmarks. To let people check GWAN's raw performances a `-b` command-line switch will disable the DoS shield.

**NB:** *G-WAN/Linux performances do not suffer from the DoS protection system.*



## THE DEVIL IS IN THE DETAILS

Accoria Rock web server is the official 2008/2009 SpecWeb champion. Given the competition, Rock deserves the respect it has won (the hard way).

But Rock has some *by-design* weaknesses. For example, HTTP 404 errors (not found) are *4 times slower* than the HTTP requests served by Rock's cache.

By contrast, G-WAN's 404 errors are slightly *faster* than cached HTTP requests (*and G-WAN without its cache is faster than Rock with its cache*).

Why does it matter? After all, HTTP errors only represent a small fraction of all HTTP requests. True, but they are not always evenly distributed. If they come in bursts (like this is the case during attacks) then speed may very well make the difference between success and failure to continue serving legit users.

The same goes for dynamic contents. G-WAN ANSI C scripts are slightly *faster* than cached HTTP requests (that's *not* the case for ASP.Net, Java, PHP, etc.).

G-WAN (on a single machine) passed the "*Slashdot effect*" without a glitch -despite a massive DoS attack that started 24h *before* the Slashdot article.

Would have G-WAN been 4 times slower it would have made no difference because the traffic was too low to saturate G-WAN -but many other web sites either have more visitors or more content to distribute than gwan.ch (or both).

A server is exposed to the public. It can't afford to fail. For this to happen, you have to make *everything* it does work as fast as possible.

G-WAN already delivers this -for free, but more is to come soon as TrusLeap's goal is not merely to offer better alternatives.



# I. The web server

## Installation and startup

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

Windows tip: after 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 by modifying Registry entries (old values are renamed).

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

C:\GWAN\www      where you will copy HTML files and image files  
C:\GWAN\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 are 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 (no parameters, will listen on all interfaces on port 80)
            gwan -listen:*.443
            gwan -listen:192.168.7.8:443
```

Listeners will not use CPU resources if no incoming connection comes.

## Log files

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



G-WAN's performances are only slightly lower when log files are enabled. The difference is negligible.

You have to stop and restart G-WAN to apply your log files changes.

### 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, If-None-Match, If-Not-Match (Etags)
Others	cache updated in real-time, directory listings, real-time servlets.

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

### Supported MIME types

"xls", "application/excel"  
"tgz", "application/x-tar-gz"  
"tar", "application/x-tar"  
"gz", "application/x-gunzip"  
"arj", "application/x-arj-compressed"  
"rar", "application/x-arj-compressed"  
"mp3", "audio/mpeg"  
"wav", "audio/wav"  
"avi", "video/x-msvideo"  
"mov", "video/quicktime"  
"flv", "video/x-flv"  
"mng", "video/x-mng"  
"mpeg", "video/mpeg"  
"mpg", "video/mpeg"  
"asx", "video/x-ms-asf"  
"wmv", "video/x-ms-wmv"  
"bin", "application/octet-stream"  
"exe", "application/octet-stream"  
"dll", "application/octet-stream"  
"swf", "application/x-shockwave-flash"  
"der", "application/x-x509-ca-cert"  
"pem", "application/x-x509-ca-cert"  
"crt", "application/x-x509-ca-cert"  
"ps", "application/postscript"  
"eps", "application/postscript"  
"ai", "application/postscript"  
"js", "application/x-javascript"



```
"xml", "application/xml"  
"json", "application/json"  
"atom", "application/atom+xml"  
"rss", "application/rss+xml"  
"rtf", "text/richtext"  
"gwan", "application/global-wan"  
"txt", "text/plain"  
"zip", "application/octet-stream"  
"pdf", "application/pdf"  
"tif", "image/tiff"  
"ico", "image/x-icon"  
"bmp", "image/x-ms-bmp"  
"svg", "image/svg+xml"  
"css", "text/css"  
"jpeg", "image/jpeg"  
"jpg", "image/jpeg"  
"png", "image/png"  
"gif", "image/gif"  
"shtm", "text/html"  
"shtml", "text/html"  
"html", "text/html"  
"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.

### **Updating static contents**

When you want to update documents located in the *www* directory you can do so without stopping G-WAN (cached documents are reloaded in real-time).

### **Default style sheet and HTTP Errors**

To personalize the HTTP default style sheet (also used by directory listings), you have to make your CSS style available under */www/imgs/style.css*.

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

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



## II. Dynamic contents

Servers need *scripts* for rapid-development and *compiled filters* for raw speed. G-WAN's ANSI C89 scripts do both -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.

If you learn, there is no other language with as much code available than C. And C is easier to learn (32 keywords, 146 functions) and faster than all others.

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 that 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 (the "servlets").

C servlets are executed by G-WAN when client request 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. All the information necessary for a redirect is in the headers. The body of the response is typically empty, but one is created here to see how to proceed:

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

    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='\"new.html\">here</a>.</body></html>");

    // reply may have changed if more memory was allocated during formatting
    set_reply(argv, &reply); // so we update it here
    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, 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 quickly 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 routines will help you in the task of building a reply.



## 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 immensely 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 dynamic buffer that you have created (but *never* free the server "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_replfrto()</code>	like the call above, but from/to given pointers in the buffer
<code>xbuf_free()</code>	release the memory previously allocated for a dynamic 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>	encode an URL so you can use it
<code>escape_html()</code>	encode a buffer so you can use it in HTML
<code>unescape_html()</code>	decode a buffer
<code>html2txt()</code>	remove all HTML tags from a buffer
<code>s_sprintf()</code>	equivalent to the C runtime call; but safer and faster
<code>s_time()</code>	equivalent to <code>time(0)</code> (but faster under Windows)
<code>s_gmtime()</code>	equivalent to <code>gmtime()</code> ; but faster (and thread-safe)
<code>s_asctime()</code>	equivalent to <code>asctime()</code> ; but faster (and thread-safe)
<code>s_localtime()</code>	equivalent to <code>localtime()</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 whole HTML page.

When users press the 'Calculate' button, the loan is displayed in the same page used to gather data entered by the end-user.

This example can be used as the basis of more complex Web 2.0 applications (SQL libraries provides *persistence* for session handling).

Another advantage of using AJAX here is the fact that it saves you from using `mod_rewrite` to have “pretty” URLs while taking advantage of the server-side functionality of servlets. All search engines heavily visited G-WAN's `csp?` URLs.



## 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 288,796 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 requires a lot of time to implement).

PHP and Python *cannot* perform well because they are *not* thread-safe. Java is



slower than ASP.Net and ASP.Net C# on IIS 7.0 is 5 times slower than G-WAN.

To benchmark such a servlet you have to time the C script execution time but also the server processing and *reply time* which can be calculated by Apache Benchmark with different concurrency loads (-c 10, 100, 500, 1000):

```
ab -n 10000 -c 10 -t 1
http://10.10.2.4:80/csp?loan&name=-&amount= 1000000&rate=3.5&term=10
```

### 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 resource is not found (404). 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 which can easily be pinpointed:



- the surface of vulnerability is expanding with new Web 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 this 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 at all times).

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.

And, as G-WAN illustrated it, there is also room for improvement in this matter.

Money buys time -and time is needed to do good things. Any technical help (hardware and software for other operating systems) or financial help (engineers also need to be paid) will be welcome and used with the efficiency you can already see today.



### III. Extending the Joy

A development platform *must* let developers and third-parties extend its features. And this must be as easy as possible -both to save time and to avoid errors.

G-WAN works with *Sservlets* (to handle HTTP forms, query databases, etc.), *Handlers* (to filter, encode, authenticate, log, etc.), *Libraries* (to add new functions to Servlets and Handlers) and *Applets* (on the client side).

#### A word about interfaces

Usually, Plug-ins connect with the server through *interfaces*. They rely on formats that you have to learn, they are uselessly error-prone and complex -and they can even become obsolete and replaced (it was the case for Microsoft IIS).

For all those reasons, our belief is that the best interface is *none*:

- Servlets copied into the /csp sub-folder will be used;
- Handlers copied into the /handlers sub-folder will be used;
- Libraries copied into the /libraries sub-folder will be used.

To disable a servlet/handler/library just delete or rename it extension (to \*.cx, \*.dlx -or anything else than the expected \*.c and \*.dll or \*.so).

To disable these capabilities, remove the /csp sub-folder to completely disable *servlets*, the /handlers sub-folder to completely disable *Handlers* and the /libraries sub-folder to completely disable *Libraries*.

You do not have to stop and restart gwan to update modified servlets (or to load new ones). G-WAN does it on-the-fly.

#### Servlets

*Servlets* are covered in details in Chapter II.

#### Handlers

*Handlers* can either be C source code (like servlets) or \*.dll/\*.so like *Libraries*. A future version of G-WAN will provide two C source code *Handler* samples:

- 'ip2geo.c' (filtering incoming connections by country),
- 'syslog.c' (to log connections at a remote syslog server).

A good *Handler* compiled (DLL) sample will be the G-WAN SSL.DLL.



## **Libraries**

Third-party *Libraries* extend the features made available to C servlets.

The 'sql.c' servlet sample is provided for the SQLite library version 3.  
The 'mandel.c' servlet sample illustrates the use of the Boutell GD library.

G-WAN lets you use the GD library to create dynamic pictures, the GNU GSL library for scientific calculations, the Crypto library of your choice, and so on.

Any open-source, commercial, system or custom-made DLL written in your favorite language can be used by G-WAN *without modification nor dedicated interfaces*.

## **Applets**

*Applets* will be illustrated at a later date when real-life examples will be available.

The purpose of G-WAN applets is to give the client-side as much power as you can already find at the G-WAN server-side.



## 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.



## Usage Terms and Conditions

(no cryptic wording)

These terms and conditions govern the distribution, licensing and delivery of the G-WAN software product to you by TrustLeap.

BY ACCEPTING DELIVERY OF THE PRODUCTS AND SERVICES DESCRIBED ON TRUSTLEAP'S WEB SITE OR OTHER TRUSTLEAP'S DOCUMENTATION, USER AGREES TO BE BOUND BY AND ACCEPTS THESE TERMS AND CONDITIONS OF USE UNLESS CUSTOMER AND TRUSTLEAP HAVE SIGNED A SEPARATE AGREEMENT, IN WHICH CASE THE SEPARATE AGREEMENT WILL GOVERN.

These Terms constitute a binding contract between user and TrustLeap. User acknowledges agreement and acceptance of these Terms by making installing the software. These Terms may change without prior notice.

We reserve the right to make adjustments to distribution terms, products and service offerings for reasons including, but not limited to, changing market conditions. We make every effort to ensure the accuracy of the information available on our web site.

However, the documents and graphics published on this site may contain technical inaccuracies or typographical errors. We make no representations about the suitability of the information and graphics presented on this site. All such documents and graphics are provided "as is" without warranty of any kind.

### Restrictions on Use

"Modifications" means any addition, alteration or deletion from the original files distributed by TrustLeap.

TrustLeap hereby grants you a world-wide, royalty-free, non-exclusive license to copy and distribute the binary code versions of G-WAN at the condition no Modifications are made either to the files made available to you by TrustLeap.

"Response header" is the part of the response message output by the G-WAN Web server, containing but not limited to, header fields for date, content-type, server identification and cache control.

"Server identification field" means the field in the response header which contains the text "Server: G-WAN/x.x.x" where "x.x.x" is the program version number.

You agree not to remove or modify the server identification field contained in the response header.



In consideration for the licenses granted by TrustLeap to you herein, you may also promote G-WAN by displaying the G-WAN 'powered' logo ([http://gwan.com/imgs/trustleap\\_pw.gif](http://gwan.com/imgs/trustleap_pw.gif)) in marketing and promotional materials such as the home page of your Web site, other Web pages or any other document. You also agree that TrustLeap may identify your company as a user of G-WAN in conjunction with its own marketing efforts.

This agreement will terminate immediately and without further notice if you fail to comply with any provision of this agreement. Upon termination, You agree to uninstall or destroy all copies of the G-WAN files.

#### Distribution

Provided that you do not modify the files of the distribution archive you are hereby licensed to make as many copies of the software and documentation as you wish; give exact copies of the original version to anyone; and distribute the software and documentation in its unmodified form via electronic means.

There is no charge for any of the above.

You are specifically prohibited from charging, or requesting donations, for any such copies, however made.

#### Obtaining the Latest Version

Before distributing G-WAN please verify that you have the latest version. The latest version is always available on our website <http://trustleap.ch/>

The filename for the G-WAN distribution archive is: gwan-bin.zip

#### Suggested One Line Program Description

G-WAN is a Web application server with 'edit & play' C servlets

#### Suggested Description

G-WAN is a Web server and an application server with C servlets and the whole takes 100 KB of code in addition to be far faster than other available Web server.

C servlets are 'edit & play' scripts that let you use the power of C with the convenience of scripts.

G-WAN is free for all. Feel free to distribute it around you!!!

#### Requirements

G-WAN Requires Windows XP (or more recent), or Linux (Ubuntu 8+)



## Copyright notice

(c) Copyright 2009 TrustLeap, All Rights Reserved.

Other mentioned trademarks and brands are the property of their respective owners. Additional copyright notices and license terms applicable to portions of the Software are set forth in <http://trustleap.ch/thirdpartylicenses.pdf>.

## 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 LICENSING FOR SUCH PRODUCTS, TRUSTLEAP ASSUMES NO LIABILITY WHATSOEVER, AND TRUSTLEAP DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO LICENSING 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.