Technical analysis of the Ultrasurf proxying software

Appelbaum, J.R.

Published: 01/01/2012

Document Version
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Technical analysis of the Ultrasurf proxying software

Jacob Appelbaum
jacob@torproject.org
The Tor Project

Abstract

Ultrasurf is a proxy-based program promoted for Internet censorship circumvention. This report gives a technical analysis of the Ultrasurf software and network. We present the results of reverse engineering the Ultrasurf client program, give an in-depth study of the known Ultrasurf network, especially those portions that interface in some way with the client or the Internet, and discuss network signatures that would allow an adversary to detect its use on a network. We cover client bootstrapping methods, censorship and censorship resistance, anonymity, user tagging by Ultrasurf and other parties, cryptographic internals and other previously unknown or undiscovered details about the Ultrasurf client and the Ultrasurf network. We find that it is possible to monitor and block the use of Ultrasurf using commercial off-the-shelf software. In particular, BlueCoat sells software and hardware solutions with such capabilities that have been deployed in Syria and other countries.

The vulnerabilities presented in this paper are not merely theoretical in nature; they may present life-threatening danger in hostile situations. We recommend against the use of Ultrasurf for anonymity, security, privacy and Internet censorship circumvention.

1 Introduction

This paper provides an in-depth study of Ultrasurf, a piece of software produced by UltraReach Inc. —a program promoted for Internet censorship circumvention, security, privacy, and anonymity. Ultrasurf is part of a broader ecosystem of circumvention software tools that attempt to thwart traffic analysis, resist Internet censorship and promote anonymity online. Ultrasurf has been promoted by various groups as an effective, safe, privacy oriented, security, anonymity and censorship circumvention tool [35, 36, 37, 39]. There is a large body of work in this field [45] and currently Ultrasurf is almost entirely missing from the literature. Previous attempts at analysis [32, 34, 41, 42] did not look in-depth at the Ultrasurf network or technical architecture. Other audits have suggested that Ultrasurf is malware [17, 18, 41], a backdoor or otherwise unsafe [26, 41] to use.

The Berkman Center for Internet & Society's 2007 Circumvention Landscape Report released in 2009 [36] stated the following about Ultrasurf and UltraReach: *UltraReach can be recommended for widespread use as the best performing of all the tested tools, though users concerned about anonymity should be warned to disable browser support for active content.* The Berkman Center's recommendation is based primarily on subjective perception of latency and throughput. We call out this report specifically as it is often cited as a blanket endorsement of UltraReach's claims about Ultrasurf. While the report did look at security at a very high level, it essentially categorized all security issues as anonymity or proxy bypass issues. As a result, it did not uncover or identify most of the serious issues present in Ultrasurf. In this paper we perform a deeper analysis of the technical architecture and discover several serious security problems.

1.1 Security and anonymity claims

This paper addresses the following claims [27, 28, 29, 30] by UltraReach and other Ultrasurf advocates about the Ultrasurf client and Ultrasurf network:

1. *"Ultrasurf enables users to browse any website freely"* — refuted in Section 3.1
2. *"employs a decoying mechanism to thwart any tracing effort of its communication with its infrastructure."* — refuted in Section 5.13
3. *"Protect your privacy online with anonymous surfing and browsing. Ultrasurf hides your IP address, clears browsing history, cookies, and more."* — refuted in Section 6.2 and Section 6.3.
4. *"change IP addresses a million times an hour"* — refuted in Section 6.1
5. *"Untraceable"* — refuted in Section 6.10
6. *"Unblockable: Client uses wide array of discovery mechanisms to find an available proxy server and, when necessary, to switch/hop to avoid tracking/blocking"* — refuted in Section 6.8
7. *"Invisible: Leaves no traces on the user's computer, and its traffic is indistinguishable from normal access to HTTPS sites"* — refuted in Section 5.12
8. *"Anonymous: No registration is requires [sic], and no personally identifying information collected"* — refuted in Section 6.10
9. *"Tamperproof: Using privately-signed SSL certificates which dont depend on external, potentially compromised CAs (thus preempting MITM attacks), Ultrasurf proactively detects attempts by censors to reverse-engineer, sabotage, or otherwise interfere in the secure operation of the tool"* — refuted in Section 5.8.

We conclude that each of these claims is false, incorrect, or misleading. We also conclude that Ultrasurf meets many, if not all, of the commonly accepted Snake–Oil [46] criteria.

1.2 Methodology

Ultrasurf is primarily protected by security-through-obscurity techniques. This method of protection is well regarded as nearly worthless if it is the primary method of protection. The security economics of analysis by reverse engineering generally lead a novice to think that security-through-obscurity will stop everyone from understanding how a given system works. Generally, the security community understands that the real strength of the system must be the design of the system itself, and not in obscuring how the system itself works. An attacker such as a government has ample resources and it is incentivized to attack tools it finds interesting.

While they are time-consuming to research, we ultimately believe the techniques used by Ultrasurf are severely flawed. This audit was performed with limited time and a limited budget by one person. A censor or dedicated attacker will not be as limited and they will likely be much more skilled with Windows reverse engineering than the authors of this paper.

Obfuscation and secrecy impede researchers, users, and advocates more than they impede most adversaries. Many adversaries have very specific motivations and such adversaries may be willing to engage in unlawful activity that the author of this paper is unwilling to engage in. They may even choose to exploit backdoors that are well intentioned and useful for so-called lawful interception. They may choose to exploit these vulnerabilities technically or socially.

We reverse engineered the Ultrasurf client with the help of Wireshark for network traffic analysis, Ida Pro with Hex Rays for static binary analysis, Wine for Win32 API emulation, GNU GDB for debugging, as well as VMWare Workstation, git, GNU GCC, Python, strace, ltrace, and nmap. Additionally, we used many different versions of Ultrasurf including the most recent release as of November 26th, 2011 – version 10.17. A full list of Ultrasurf binaries may be found in Appendix D. Run time and static analysis was performed on Ubuntu 11.04 and Windows 7. This document was prepared with LaTeX version LaTeX 2_ε. When possible Free Software tools were used to encourage independent reproduction of the claims made in this paper.

Most of this research was done while traveling in Brazil, Canada, Germany, and very small amount of it was performed in the US. Additionally, a number of interesting data points come from interception devices in Syria. As

of early April 2012, an independent tester confirmed many of the findings in this paper from China; the versions of Ultrasurf tested did directly connect to blocked addresses and did not in-fact work at all. Newer versions appear to have different, not yet blocked, addresses baked into the program.

The Ultrasurf client uses anti-debugging techniques to prevent dynamic analysis at run time. We were able to bypass these techniques and inspect the Ultrasurf client while it runs. Additionally, the client uses obfuscation techniques that are collectively known as binary packing [1, 2] as a method of preventing static and dynamic binary analysis. We were able to bypass most of the obfuscation with a combination of manual and automatic unpacking. Time did not permit for a full disassembly and full decompilation but such an endeavor is no doubt possible.

Reverse-engineering makes some conclusions more tentative than they would otherwise be and so we encourage UltraReach to publish a response. We especially encourage the publication of their source code, design and architectural documents, data retention policies, and other related facts to help clarify the issues discussed in this paper. We believe everything in this paper is factually correct and supported by evidence gathered during analysis.

2 Ultrasurf architecture overview

Although the internal structure of an Ultrasurf server consists of many layers of network proxy software, the server is effectively a single hop proxy and the Ultrasurf network is essentially a single entity; though we find that while UltraReach appears to control the network, they are merely customers of other entities. There are many scenarios where an attacker is able to compromise a single part of the server or network infrastructure. Such a compromise is almost always enough to effectively cancel out any protection that such a system may offer.

Single hop or single entity proxy [3] systems such as Anonymizer, SafeWeb, Ultrasurf, and other simple proxy servers are vulnerable to myriad issues:

- A proxy server may be compromised by an attacker.
- The proxy system or service may be run by an untrustworthy party.
- The server or proxy system may be trusted but the servers network may be monitored by an attacker.

The Ultrasurf client appears to contain two slightly novel features. The first is a customized cryptographic handshake for the transport protocol between Ultrasurf clients and a given Ultrasurf server; we discuss it further in Section 5.9. The second is the use of special DNS requests for bootstrapping knowledge about new Ultrasurf servers; we discuss it further in Section 5.6.

2.1 Connecting through the Ultrasurf network

Users have no knowledge about the actual Ultrasurf network or any of the bootstrapping processes. Ultrasurf does not publish descriptions of the actual Ultrasurf network or any of the bootstrapping processes. At initial run time the Ultrasurf client will connect directly to a list of IP addresses that we assume are Ultrasurf servers. In practice it appears that initial access to the Ultrasurf network requires at least a single TCP connection to a host whose address is embedded in an Ultrasurf client. This connection appears to be encrypted and UltraReach claims that this protocol is SSL [27]. Our observations suggest that this is accurate, but not the entire story. The Ultrasurf client creates a local HTTP proxy upon successfully connecting to any Ultrasurf server; this proxy is discussed in Section 5.16.

Access to the Internet through the Ultrasurf network is only possible by interfacing with this local HTTP proxy. Any connections that are sent through the proxy will be relayed to the Internet through a series of filtering proxies running on the selected Ultrasurf server. The Ultrasurf servers generally bind traffic bidirectionally to a single IP address. That is, when an Ultrasurf client connects to an Ultrasurf server, all connections appear to originate from the Ultrasurf servers IP address. Subsequently the Ultrasurf client will attempt to use DNS queries in addition to TCP connections for bootstrapping server information about the Ultrasurf network.

3 The Ultrasurf Network

The Ultrasurf network is directly comprised of a handful of Ultrasurf servers; overall the network contains dozens of static IP addresses across a few large network blocks. During the discovery phase as described in Section 5.4, the Ultrasurf client will find servers and cache them on the local disk.

```
65.49.14.0/24
111.255.176.0/24
```

Figure 1: The core IP blocks for the Ultrasurf network

```
65.49.14.88:443
65.49.14.87:443
65.49.14.79:443
111.254.49.188:443
220.131.214.80:443
1.174.1.123:443
124.12.58.192:443
175.182.21.135:443
59.115.245.40:443
```

Figure 2: List of servers from a selected run

The Ultrasurf network may also contact authoritative DNS servers. These DNS servers are running ISC BIND DNS server software as noted in Section 6.5. The Ultrasurf client does not generally directly contact those DNS servers. These servers are contacted indirectly by Ultrasurf clients through the Ultrasurf network. Each client ships with a list of open recursive DNS servers that are embedded inside each Ultrasurf client. The recursive DNS servers are not strictly part of the Ultrasurf network; the recursive servers appear to not be run by UltraReach while the authoritative DNS servers are operated by UltraReach. It seems plausible that UltraReach simply found a list of open recursive DNS servers and added that list to the Ultrasurf client binary.

The Ultrasurf network as presented by the current Ultrasurf client appears to be a very simple single hop HTTP proxy as described in Section 5.16. The IP address that the Ultrasurf client uses for entrance to the Ultrasurf network appears to be the same as the IP address that remote servers will see as the client's IP address. In practice access to the Ultrasurf network requires at least a single TCP connection to a host that is embedded in the Ultrasurf client. The Ultrasurf client will attempt to use DNS queries in addition to TCP connections for bootstrapping but not for transport of data. The Ultrasurf network appears to block access to some pages as a matter of policy. This filtering is discussed in the following Section 3.1.

3.1 Network censorship: New boss, same as the old boss

The Ultrasurf network is comprised of one or more Ultrasurf servers. Each Ultrasurf server proxies all client connections through a series of local proxy servers; this is generally referred to as a chained proxy or a proxy chain. It appears that the user directly interfaces with a Squid proxy [4] and that the Squid proxy interfaces with a ziproxy [5]. The ziproxy in turn directly talks to sites on the Internet. Each proxy on the Ultrasurf server is configured with an access control list (ACL) that prevents access to certain sites or systems. The proxy systems collect extensive log [6] entries. When the ACL prohibits a specific site, the user is redirected to a block page as shown in Figure 38. The server appears to log this information [6] in addition to setting an HTTP cookie for the Google Analytics service on the actual block page itself. Technically, when a blocked site is encountered, Ultrasurf's servers return a 302 redirect in response to the respective CONNECT/GET request as shown in Figure 3. The Squid errors additionally provide enough information

to act as a censorship oracle. This means that it is possible to extract a list of all sites that are censored by the Ultrasurf network or otherwise unavailable for other reasons. While we found only artistic sites blocked by this filter, we did not extract the entire censorship list; such extraction is trivial and is left as an exercise to the reader. None of the blocked sites were known to be illegal and while adult content was involved, many legal US based websites and companies were censored seemingly based on their content, rather than their legal standing.

```
HTTP/1.0 302 Moved Temporarily
Server: squid/2.7.STABLE7
Date: Wed, 14 Sep 2011 22:07:33 GMT
Content-Length: 0
Location: http://www.ultrareach.com//block.htm
Connection: keep-alive
Proxy-Connection: keep-alive
```

Figure 3: Blocked pages are redirected with HTTP 302

The GET request in Figure 4 is intercepted by the ziproxy rather than the Squid proxy.

```
HTTP GET http://ossipee.cs.dartmouth.edu:8008/ HTTP/1.1

HTTP/1.0 200 OK
Server: ziproxy
Date: Wed, 14 Sep 2011 23:06:04 GMT
Content-Type: text/html
Connection: close
Content-Length: 452
```

Figure 4: ziproxy HTTP 200 OK

While the ziproxy has intercepted the request the remote peer sees the data shown in Figure 5.

```
GET / HTTP/1.0
Host: ossipee.cs.dartmouth.edu:8008
User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:6.0.2) Gecko/20100101
Firefox/6.0.2
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
DNT: 1
Cache-Control: max-age=2419200
Accept-Encoding: gzip
Connection: close
```

Figure 5: Information leaked to remote non-Ultrasurf server

The connection is closed as expected by the above client declaration. The data in Figure 6 is one of the lines returned to the user.

```
<meta http-equiv="REFRESH" content="0;url=http://ultra\_error">
```

Figure 6: Errors with the proxy return a metarefresh tag

The data returned suggests that the ziproxy and the Squid proxy are not perfectly harmonized in their configuration. Additionally, it appears that a very well-known criticism website [7] of the Falun Gong is unreachable through Ultrasurf as seen in Figure 7. It is rumored that this website is run by the Chinese government.

```
GET http://www.facts.org.cn/ HTTP/1.1

HTTP/1.0 504 Gateway Time-out  (text/html)

HTTP CONNECT www.facts.org.cn:443 HTTP/1.1

HTTP/1.0 504 Gateway Time-out
Server: squid/2.7.STABLE7
Date: Wed, 14 Sep 2011 22:23:49 GMT
Content-Type: text/html
Content-Length: 906
X-Squid-Error: ERR_CONNECT_FAIL 110
```

Figure 7: Filtering by the Great Firewall blocks access from the Ultrasurf network

It may be that the so called Great Firewall of China is blocking connections from known nodes in the Ultrasurf network. DNS appears to be functional as the error is related to connecting, rather than the *X-Squid-Error: ERR_DNS_FAIL* we would expect from a DNS related failure.

4 The Ultrasurf Server

4.1 A typical Ultrasurf server

An Ultrasurf server is a system that any Ultrasurf client may use as a single hop proxy. Each Ultrasurf server appears to have a common host name “local” or “linux” and each server seems to share a common list of services amongst most Ultrasurf servers. Generally, an Ultrasurf server has the TCP ports open as listed in Figure 8.

TCP port	Service description	Service note
80	HTTP	Varied web server software
443	TLS	Custom SSL/TLS service
554	RTSP	Unconfirmed as RTSP
1723	PPTP	Leaks platform and hostname

Figure 8: Open TCP ports on a typical Ultrasurf server

The web server software detected included squid, lighttpd, Apache httpd, Microsoft Windows Media Server and other unknown web servers. Specific versions are enumerated in the section 6.5. Each Ultrasurf server also generally appears to accept the IP protocols listed in Figure 9.

IP protocol	Protocol name	Protocol note
1	ICMP	Timing information
6	TCP	See Figure 8
17	UDP	DNS
47	GRE	VPN related protocol

Figure 9: IP protocols accepted by a typical Ultrasurf server

4.2 Timing is everything

It is possible to remotely read the system time from three services - the first method is by reading the HTTP date on the web server running on TCP port 80. The second method is by inspecting the Ultrasurf SSL/TLS handshake in detail. The third method is by inspecting the proxy headers from the proxy offered by the Ultrasurf client. These methods have been implemented by the TeaTime [33] utility for this paper. This timing measurement is extremely useful for understanding the remote network and service topology, detecting possible network address translation (NAT), and for confirming that each of these services does indeed run on a single common system.

4.3 Proxy Turtle chains all the way down

As discussed in Section 3.1, each Ultrasurf server has a number of proxies running that are chained together. Each proxy has awareness of special host names and each has ACL restrictions that are unique. Figure 10 shows an incomplete list of special host names.

```
local
localhost
ziproxy
ultra
ultra_error
```

Figure 10: Special host names available through the Ultrasurf client proxy

Generally, each server appears to contain a Squid [4] proxy and a ziproxy [5] in a proxy chain. Additionally, it appears that each server runs a web page proxy that is running on a web server on the local network interface as seen in Figure 11.

```
http://localhost:8080/001/www.ntdtv.com
```

Figure 11: web proxy apparently running on the Ultrasurf server

The above URL is only reachable through the Ultrasurf client's local proxy as described in Section 5.16. It allows for multiple loops to be triggered internally with the proxy chaining when passed a simple URL as seen in Figure 12. Requesting such URLs will sometimes cause encoding and decoding issues related to gzip.

```
http://127.0.0.1:8080/001/127.0.0.1:8080/001/127.0.0.1:8080/001/127.0.0.1:8080/001/
```

Figure 12: Proxy chaining with GET request

This proxy software may be Psiphon [8] or CGIProxy [9] though neither is confirmed. It appears to be Psiphon [21] based on the URL construction [22] but this is merely a guess. It appears that it is possible to partially bypass the Squid ACL restrictions using the proxy software. HTTP CONNECT requests to 127.0.0.1 are forbidden while GET requests to 127.0.0.1 are allowed. The attack surface created by such a web application proxy is non-trivial.

4.4 Ultrasurf server DNS resolution

Each Ultrasurf server performs at least two DNS lookups per host name; this artifact appears to be related to the proxy chain configuration. When attempting to connect to a host without a web server as a method of forcing a DNS lookup, we see results similar to this in Figure 13.

```
00:20:42.189296 IP 65.19.175.2.38356 > 13.13.206.254.53: 38440 A?
174-37-205-87.robot.example-redacted.com. (49)
00:20:42.190273 IP 65.19.175.2.52549 > 13.13.206.254.53: 51551 AAAA?
174-37-205-87.robot.example-redacted.com. (49)
```

Figure 13: DNS queries from the Ultrasurf network to authoritative name servers. The actual domain name has been removed from this figure.

These DNS queries appear to be sent by the recursive name servers run by the Ultrasurf upstream provider. The above example is for *ns1.fmt.he.net*. and at the time, the Ultrasurf client was connected to an Ultrasurf located at *65.49.14.87:443*. This indicates that the upstream provider has both visibility and control over the Ultrasurf clients in a very powerful fashion. It also shows that Ultrasurf's software, which does not otherwise appear to support IPv6, makes IPv6 related DNS queries.

5 The Ultrasurf client

5.1 Downloading the Ultrasurf client

The Ultrasurf client is available for download from the main UltraReach servers only over insecure channels as none of the UltraReach download sites offer HTTPS. MD5 hashes of some versions of the Ultrasurf client are offered but only over the same insecure HTTP servers; a similar page with hashes is offered as an internal URL as seen in Figure 39. UltraReach further encourages users to fetch copies of Ultrasurf from any place that users might find a copy including peer to peer networks such as eMule and other file locker sites. Peer to peer networks commonly contain trojaned copies of software and encouraging users to find software in this way is generally considered poor practice. Digital signatures for specific Ultrasurf releases might mitigate such concerns; there appear to be no such signatures in any case and the hashes offered are entirely unauthenticated.

Additionally, UltraReach suggests that users email *wj@wujieliulan.com* to request a copy of Ultrasurf. Messages sent to that address in August and November of 2011 were rejected by the mail server at *wujieliulan.com*. UltraReach continues to encourage users to email for a copy of the software and are probably entirely unaware that it is a non-functional avenue for retrieval of Ultrasurf.

5.2 The Ultrasurf binary

The Ultrasurf client is a Win32 PE executable written in C++ and compiled with Visual Studio. The PE executable is packed with ExeCryptor/Themedia [1, 2] code obfuscation techniques and it makes every attempt to prevent any kind of debugging. As a result the Ultrasurf client is flagged as a virus according to many online virus and malware sample collection sites [17, 18]. It is impossible for the user to know if this warning is a false positive. If an adversary were to replace or compromise an Ultrasurf binary, it would be impossible to know if a backdoor was inserted as it would

likely trigger the same malware or virus heuristics. In some cases, virus and malware software detects that the binary is Ultrasurf and decides that it is safe; this kind of analysis may result in a false negative that puts a user in harm's way.

The Ultrasurf client attempts to load shared libraries directly or indirectly as listed in Appendix A The list of imported functions varies slightly based on version and not all functions are actually called for each run.

5.3 Running the Ultrasurf client

The Ultrasurf client is designed for use on Windows and uses the Win32 API. It is also possible to run the Ultrasurf client under Wine on Gnu/Linux platforms. The Ultrasurf client claims to be an "Install free" program but this is not strictly correct. The Ultrasurf client modifies the local registry and it writes multiple files to the file system. It also changes the local Windows Cookie and Temporary Internet Files directory. The Ultrasurf client caches network information in a local directory that is relative to wherever the binary is run. Additionally, a text configuration file "u.ini" is written to the local disk as well.

The Ultrasurf client automatically performs network discovery of possible Ultrasurf servers to use as described in Section 5.4. It caches discovered Ultrasurf server information locally and offers a local HTTP proxy as described in Section 5.16; it may bind to other local UDP ports if it is performing DNS bootstrapping. It is possible to configure the Ultrasurf client to use a proxy to reach the Internet if a direct connection is unavailable.

5.4 Ultrasurf client network discovery bootstrapping

Ultrasurf uses a multi-stage bootstrap process to learn about and connect to servers in the Ultrasurf network. It also generates supposedly "normal" HTTPS traffic as a kind of chaff for an as of yet unknown list of domain names at various predictable times during the bootstrapping process.

The first stage of the bootstrapping process occurs when the Ultrasurf client attempts to connect to a list of servers generally found on TCP port 443. This appears to be a non-standard SSL/TLS service. For the 10.x family, the Ultrasurf client sends three TCP SYN packets to hosts in the 65.49.14.0/24 network block. If these are blocked, Ultrasurf may simply fail outright. If they are not blocked, Ultrasurf will cache some information in a local directory (e.g.: *utmp/Gmamewmmwymh10d6f*). This local cache is referenced internally by the Ultrasurf client as the *STATICCache*. This network information appears to come from the information retrieved as part of the update process described in Section 35.

The second stage is via DNS queries. This local cache is referenced internally by the Ultrasurf client as the *DNSCache*. The domains used in the queries are referenced and stored in the *DOMAINCache*. It appears that the Ultrasurf client does not embed DNS names such as *dwvrl.info*, rather those second level domain names appear to be fetched from Ultrasurf servers on a regular basis. The DNS queries appear to follow a time based pattern for the construction of the third and fourth level domain names such as *doau.vxfexfez.dwvrl.info*.

The third stage attempts to fetch web pages that contain so called *"PGP"* messages. This information is locally cached and referenced internally by the Ultrasurf client as the *Cache*.

If and when the third stage fails, Ultrasurf will fall back and attempt to connect to the hard coded list of first stage TCP services. It appears to repeat the first stage exactly and does not appear to learn from previous failures. Additionally, if the Ultrasurf client is freshly run without any local state it will make the same choices about network connections each time. A total failure to communicate with the Ultrasurf network will sometimes result in Ultrasurf crashing. Generally the UI indicates that everything is fine, that connections are working as expected and so on - even in cases of absolutely no network connectivity. The author of the paper noticed that the UI indicated perfect connectivity when the network cable was removed from the machine entirely.

5.5 Bootstrapping: Locally cached information

Ultrasurf has an embedded set of IP addresses and port numbers ("STATICCache") that it attempts to connect to upon first launch. Ultrasurf learns about new server nodes over time by connecting to Ultrasurf servers and downloading further information into a local cache.

The Ultrasurf client will cache all information discovered about the network in a directory named “utmp” that is located relative to the Ultrasurf client binary. The files inside are named in a systematic manner. The Ultrasurf client will sometimes write the same file contents to the same file name with repeated runs if state is lost or purposely reset.

A sample of collected data files from a normally running Ultrasurf client “utmp” directory is visible in Figure 14. These creates a log of every possible server a client might use and writes it to the hard disk.

```
Enikbevujkul9l0m (44 bytes)
Gmamewmmwymh9d6f (44 bytes)
Icmaamqruxrj0t2d (48 bytes)
Yahggswsaysk9w4y (48 bytes)
Eqklrkiuazud7p7g (36 bytes)
Gpcndcmmksmz7h3h (36 bytes)
Ifobqzdrlmrb8x7x (36 bytes)
Ydjqffjsrssc7a1r (36 bytes)
```

Figure 14: Files created by Ultrasurf in the local *utmp* directory

While there are obvious patterns such as the repeating hex bytes of *ba befa* in the data as seen in Appendix B, we have not decoded the contents of any of these files. A sample hex dump for each file is visible in Figure 40. The internal code for processing these files appears to be the same code for processing all of the different data formats; after Base32 or Base64 decoding, the faux-PGP messages and DNS results are similarly structured. The internal assembler code that decodes these formats is self-contained and appears to simply right-shift bits.

In addition to the *utmp* directory and the other system changes, the Ultrasurf client will write a file named *PUTTY.RND* to the local disk. This file is 600 bytes and contains what appears to be a seed file used for random number generation. All of this data is cached on disk and while seemingly obfuscated, Ultrasurf acts as a forensic oracle to decode all of this locally logged data without needing to fully understand the encoding. This means that while some of the encoded data is not understood, it is possible to use the Ultrasurf program itself to decode the data into an understandable format. This is one of many examples that demonstrates why security through obscurity is not a reasonable security practice; it is time consuming to reverse engineer these small file formats and it is easier to simply use Ultrasurf to decode the data.

5.6 Bootstrapping: DNS

After information in the local IP cache is exhausted and when Ultrasurf has a populated DOMAINCache, the client will generate and send special DNS requests. Such a query is shown in Figure 17.

Ultrasurf embeds a list of DNS servers that open recursive queries. These are almost certainly not run by Ultrasurf but are rather used parasitically by Ultrasurf clients. If a selected server is disabled, reconfigured or no longer allows recursion, Ultrasurf will simply pick another DNS server and attempt to request recursion for an A record with a fixed DNS name such as *jfxh.bycqybwr.dwvrl.info* – Ultrasurf will attempt to contact each recursive DNS server to resolve such a name. Eventually if recursion is allowed a response will shortly follow as shown in Figure 15.

```
CNAME c4wvqbs8.ukos9q3.dwvrl.info
CNAME c4w.vqbsuko9o1xj.dwvrl.info
CNAME c4w.vqbsrlmp98pt.dwvrl.info
A 216.239.113.172
```

Figure 15: CNAME response example

Internally, each of these returned CNAME records translates to a single Ultrasurf server, port number and other pieces of information. The bit width of the second and third level domains appears to be statically fixed at fifteen bytes when excluding “.” from the count.

The Ultrasurf bootstrapping process uses a kind of slow-flux DNS discovery process. The reason that we choose the term slow-flux rather than the more common fast-flux name is literally the speed at which it happens.

Each Ultrasurf client has a cache of recursive name servers that it will use before falling back to a simple `gethostbyname()` DNS resolution process that uses the local system's resolver. Tracking or blocking Ultrasurf DNS queries seems straightforward, while the domains change often, it appears that the domains are only *.info* domains. Additionally, the second and third level names from the first byte until the second '.' are always sixteen characters in length when the final '.' is not counted. While the second and first level domains regularly change, it seems unlikely that the bit width will change. Thus it seems likely that the DNS bootstrapping method has a rather unique signature.

The DNS packets in question do not appear to be encoded with a standard DNS tunneling framework. It appears to be a custom encoding written by Ultrasurf that uses Base32 symbols. If it is encrypted, I expect it is done with some kind of shared symmetric key embedded in the binary and that some bits of the query change over time.

The first run of Ultrasurf will not perform any DNS queries. Only after subsequent runs will the Ultrasurf client attempt to fetch information via DNS resolution. It appears that the first run of Ultrasurf fetches node information and caches it to disk in the local *utmp* directory.

When the DomainCache is populated, the Ultrasurf client will read from the local domain cache file and decode possible domains to query as shown in Figure 16.

```
Load 1 from DOMAINCache
Load DOMAINCache: DWVRL.INFO
```

Figure 16: DOMAINCache loading log

Further DNS queries will be made for each domain in the DOMAINCache. The top level domain *.info* appears to be irrelevant. The client queries for a fixed host per domain in the cache during a given window of time until it finds a response. Thus upon restoration of previous client state, an Ultrasurf client will make a query for *doau.vxfexfez.dwvrl.info* twice or as many times as the state is reset.

The normal process for discovery for the *dwvrl.info* domain is as follows. The Ultrasurf client will construct a UDP DNS query and send it as shown in Figure 17.

```
Send UDP Query (Cache) doau.vxfexfez.dwvrl.info to dns server (cache) 137.65.1.1.
```

Figure 17: DNS query

This DNS query is seen on the network as a standard A record request as shown in Figure 18.

```
Standard query A doau.vxfexfez.dwvrl.info
```

Figure 18: DNS response

The DNS server for the domain answers directly or indirectly in a single response as seen in Figure 19.

```
CNAME dcy.1371ejxt3z7z.dwvrl.info
CNAME dcy37e1j.wx4k8yq.dwvrl.info
CNAME dcy837ej9.rlcbfm.dwvrl.info
```

Figure 19: response for dwvrl.info

The bit width of all of the CNAME resources is always relatively the same size. The domain name itself is the variable length field. The first dot ('.') in the sub-domains is in a variable place but always sits between two sets of bytes.

The Ultrasurf client will then parse the reply as seen in Figure 20.

```
UDPResponse 137.65.1.1 doau.vxfexfez.dwvrl.info numRec 3
Add node (1)=112.104.13.108:443 id=1001 gp=1 ty=0 ttl=0
Skip verify Q0: 112.104.13.108
Add node (1)=114.43.193.133:443 id=1001 gp=1 ty=0 ttl=0
Skip verify Q0: 114.43.193.133
Add node (1)=125.228.238.57:443 id=1001 gp=1 ty=0 ttl=2
```

Figure 20: Parsed UDP results

The UltrasurfClient will then directly attempt to connect to the first node that is parsed from the reply as seen in Figure 21.

```
Switch to node: 0 112.104.13.108:443
```

Figure 21: Ultrasurf switching to the new node after DNS discovery

This single issue is potentially quite problematic. It appears that the network, and perhaps not the Ultrasurf network alone, has the ability to control the client's path selection process. This is discussed in detail in Section 6.4.

Other known domains include those shown in Figure 22.

```
LYYMHC.INFO
DWVRL.INFO
HXMUZ.INFO
OD4IHK9.INFO
MJXMRTF.INFO
```

Figure 22: Recently observed Ultrasurf domain names

5.7 Bootstrapping: DNS response format

As an exercise to the reader we encourage decoding subsequent replies from the server for *dwvrl.info*.

Ultrasurf sends the data shown in Figure 23.

```
1 0.000000 172.16.42.131 153.2.242.115 DNS Standard query A jfxh.bycqybwr.dwvrl.info
```

Figure 23: Client DNS query

The Ultrasurf authoritative DNS server replies with the data shown in Figure 24.

```
2 0.409178 153.2.242.115 172.16.42.131 DNS Standard query response
CNAME c4wv.qbs3tjg18oi.dwvrl.info
CNAME c4wv.qbssss9p8ah.dwvrl.info
CNAME c94.9wvqbs3tjgoi.dwvrl.info
```

Figure 24: Server DNS response

What information was sent? What information about servers nodes is encoded in the response?

The UltraReach company appears to have some kind of API for generating new domains. Personal communications suggest that they have some kind of deal with the .info domain registrar for economic reasons. It is possible that the .info domain registrar has a history of all domains previously registered by UltraReach. Such a list would allow someone to retroactively discover clients that have used the DNS bootstrapping methods. Furthermore, it is possible to construct such a list by observation of the Ultrasurf client. As each Ultrasurf client learns about each new domain name, the adversary is also able learn about it.

5.8 Bootstrapping: faux-PGP

The third stage of network discovery and bootstrapping involves fetching various web pages as part of a search for a specifically formatted message. The Ultrasurf client has internal references to several URLs that contain what is claimed to be PGP encrypted messages. These files are not any known PGP encoding and as far as we can ascertain, they are not actually PGP encrypted messages. They appear to be some kind of proprietary format that if actually encrypted, the decryption is probably tied to static keys in the Ultrasurf binary itself. The Ultrasurf client fetches these messages with the local IE wininet system library calls. These files are cached to disk in the *"Local Settings/Temporary Internet Files/Content.IE5"* directory.

Example internal Ultrasurf log lines of fetching this so called PGP message from Amazon S3 are shown in Figure 25. This directly contradicts the claims by UltraReach that they do not rely on SSL/TLS certificates issued by certificate authorities. They do not control the certificate presented by *s3.amazonaws.com*, nor who may issue that certificate.

```
Sep-15-14:13:54 |  30253 Fetching Gdoc HTTPS: https://s3.amazonaws.com/s3c001/2
Sep-15-14:13:57 |  33251 Finished Gdoc HTTPS: https://s3.amazonaws.com/s3c001/2
Sep-15-14:13:57 |  33253 Add node (3)=61.225.7.78:32561 id=1000 gp=23 ty=0 ttl=2
Sep-15-14:13:57 |  33253 Add node (3)=61.225.6.87:32561 id=1000 gp=23 ty=0 ttl=2
Sep-15-14:13:57 |  33253 Add node (3)=218.164.40.233:32561 id=1000 gp=23 ty=0 ttl=2
Sep-15-14:13:57 |  33253 Add node (3)=175.182.123.199:32561 id=1000 gp=23 ty=0 ttl=2
Sep-15-14:13:57 |  33253 Add node (3)=61.216.12.175:32561 id=1000 gp=23 ty=0 ttl=2
```

Figure 25: Ultrasurf internal log fetching and parsing the faux-PGP message

For the four known "PGP" files, we have the following character count: file "1" is 2398 bytes, file "2" is 3410 bytes, file "3" is 2030 bytes, file "4" is 1846 bytes. There are only four files on the above referenced Amazon S3 website. These files are always an even number of bytes long. They are most certainly not PGP encrypted messages but they do appear to be BASE64 encoded data. The files appear to share a common internal format with all other Ultrasurf network information cache files. This internal format is not yet fully understood but to appears to work by merely shifting bits. If there is any actual encryption, we believe that it would use static keys embedded in the binary.

In addition to fetching the faux-PGP files from Amazon's S3 server, Ultrasurf additionally fetches documents that claim to be Atom Feeds. The Ultrasurf Atom Feed as shown in Figure 26 was discovered by analysis of Blue Coat DPI logs found in Syria [31]. It led us in turn to discover this method of bootstrapping was used by the Ultrasurf client in the wild.

This analysis uncovered actual Ultrasurf users and their behavior by inspecting log files. Rather than only showing connections to a known Ultrasurf server, the log files showed lots of activity. The activity shown included, but was not limited to, attempts at bootstrapping and unproxied communications before and sometimes even after Ultrasurf had bootstrapped.

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">

<title>Atom Feed</title>
<updated>2011-10-06T02:10:59Z</updated>
<id>urn:uuid:e2e5fe28-cafc-414c-b5d0-083928f8f935</id>

<entry>
<title>Atom Feed Update</title>
<id>urn:uuid:e2e5fe28-cafc-414c-b5d0-083928f8f935</id>
<updated>2011-10-06T02:10:59Z</updated>
<content>-----BEGIN PGP MESSAGE-----
ODEyBEH7/k4GvyL6apsCab2lYgDa7Q8FXxdz2OuRtRXeZJ8p1IcdNtkmjoGRmr
8mfzjRvKZsuvAzBtGS4AQS1sxhA51Tm58DDO25N8rUuC2U0sTQ4Y4cPh/fp5TJ3GiYAhN8rvf/+Deg/hfu0X//
    BGJA6wcBpX91T
NyoQaRrFK2I+p75pLxpfL+DO13/qd+QUAqXvu8XvX6FW8BT7dghZZTAqXoRkwC0FQD3iygr7wMVhvXH/4fEybVM
-----END PGP MESSAGE------</content>
</entry>

</feed>
```

Figure 26: Ultrasurf faux-PGP embedded in ATOM feed located at
`http://65.49.14.54/Y2U0YWNkMmX5b/l2JGrVT0sm/u7bipJsXdKl/hfPx9z2dElIZ/7pj3B`

The Atom feed shown in Figure 26 represents yet another way for the Ultrasurf client to bootstrap information about the network. However, unlike the Amazon S3 method, this bootstrapping is done entirely in plain text HTTP. Additionally, it is fetched directly from the main *65.49.14.0/24* Ultrasurf network. This makes for a ripe target that appears to provide no difficult barrier for an attacker beyond understanding the message format. The URL parameters may be modified as shown in Figure 27 to generate a different faux-PGP message that appears to be treated as valid data by an Ultrasurf client.

```
<content>-----BEGIN PGP MESSAGE----- PzEyB0D7/ksbpyLga4EQabuwYwbb7gwJTBdkyvuApQPZag== -----END
    PGP MESSAGE------</content>
```

Figure 27: Modified Ultrasurf faux-PGP embedded in ATOM feed located at
`http://65.49.14.54/Y2U0YWNkMmX5b/l2JGrVT0sm/`

5.9 3-2-1 Contact

The Ultrasurf client uses a non-standard handshake to initiate client and server communication. Certain standard TLS handshake requests will elicit a proper TLS ServerHello reply; a client initiated TLS session resumption with a random session ID in a ClientHello will always receive a response. The Python code to elicit such a response is shown in Figure 28. At times an Ultrasurf client will emit a TLS ClientHello that appears to be an attempt at session resumption. The Ultrasurf server will reply with a properly formatted TLS ServerHello. It is possible to fingerprint the remote clock with the remotely echoed ServerHello. All packets after the handshake appear to be TLS records that are marked as carrying an HTTP payload. It appears that this protocol is simply a customized SSL/TLS server with handshake obfuscation to confuse normal SSL/TLS protocol parsers and classifiers.

```
#!/usr/bin/python
import socket
import binascii
sketch_host = "65.49.14.80"
sketch_port = 443
client_hello =
binascii.unhexlify("16030000610100005d03004e66376b760e6d9d0a64a7855502ea3dd7
1884e85ac61d6afc6aed3a7eb0fe3720ffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffff001600040005000a
00090064006200030006001300120063010 0")
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((sketch_host, sketch_port))
s.send(client_hello)
```

Figure 28: Python code to elicit a response from the custom SSL/TLS server

With the ClientHello sent using the code shown in Figure 28, it is possible to parse the reply and read the remote time out of the TLS ServerHello. This data as parsed by Wireshark is shown in Figure 29.

```
gmt_unix_time: Sep 14, 2011 22:55:29.000000000 CEST
```

Figure 29: Remotely detected time stamp from Ultrasurf TLS ServerHello

5.10 Client UI feedback

When the Ultrasurf client is running as a local proxy the UI allows the user to select one of three nodes as represented by a green dot per server. The GUI does not meaningfully describe anything about the servers and merely shows a green dot, a percentage number and an indicator for each of the three servers. The IP address of the nodes is not revealed to the user in the user interface. As discussed in Section 5.4, we find that this UI is misleading at best and extremely incorrect.

5.11 Successful connection to the Ultrasurf network

By default the Ultrasurf client launches an instance of Internet Explorer that loads the Ultrasurf home page with JavaScript. This JavaScript constructs tracking code that is run on each visitor's computer; these issues are discussed at length in Section 6.3. Assuming that Ultrasurf has successfully connected, it will open a local HTTP proxy on *127.0.0.1:9666*, the Ultrasurf client will check for an update as discussed in Section 5.14 and the user is free to use the local proxy. The user may use another browser such as Firefox with WJButton as discussed in Section 5.17. Microsoft's Internet Explorer will be launched after the very first run of Ultrasurf unless it is entirely unavailable. It is possible to configure Ultrasurf to behave differently in the configuration section of the application; this will not change the internal use of the Win32 API for the update subsystem or chaff traffic as discussed in Section 5.13.

5.12 Forward Secrecy and Forensics

None of the networking protocols in use appear to have any forward secrecy properties. The data used by the client is additionally written to the local disk without any regard for how this data may be used during forensics analysis at a later date.

The Ultrasurf client performs almost no practical anti-forensics. The Ultrasurf client does not have any kind of cookie isolation when used with Internet Explorer as suggested by the Ultrasurf authors. All future IE sessions may remain linkable to the state when the proxy was in use. It will leave the local configuration file and network cache on

the system. The network cache is essentially a record of which servers were used and likely when they were learned. This is essentially a log of likely servers used by the client.

Upon exit, the Ultrasurf client will close the local proxy and may attempt to set the local registry settings back to a non-proxied state. In the event of a crash the system may be left in a proxied state without a functional or running proxy. This state is not uncommon and such registry changes are non-trivial forensic markers. This directly contradicts the claims made in Section 1.1.

The Ultrasurf client leaves behind many traces on the disk. The local registry is modified with regard to fonts as well as proxy settings. Again, we find that this directly contradicts the claims made by UltraReach of being *Untraceable* and we refute that they *thwart any tracing effort.*

```
modified:   system.reg
modified:   user.reg
```

Figure 30: Both the user and system wide registry are modified

A sample of the modified registry is seen in Figure 31.

```
{-[Software\\Microsoft\\Windows\\CurrentVersion\\Fonts] 1315844275}
{+[Software\\Microsoft\\Windows\\CurrentVersion\\Fonts] 1315910420}
```

Figure 31: Example of modified registry data regarding fonts

Path names and files that are either created, modified or deleted during execution time are shown in Figure 32.

```
Desktop/utmp/
Cookies/
Local Settings/
PUTTY.RND
```

Figure 32: Areas on the local file system that change after running Ultrasurf

5.13 Client chaff

The Ultrasurf client uses Internet Explorer to generate so called “Fake HTTPS” requests. During the bootstrapping process the client connects to several HTTPS sites probably as an attempt to confuse a casual observer. The real HTTPS requests generally connect to Amazon’s AWS service and the fake, though actually HTTPS, requests go to other domains. The “PGP” message is the file fetched from the Amazon AWS page.

Example log lines are visible in Figure 33.

```
Sep-15-14:13:47 |  22764 Fetching Fake HTTPS: https://share.avvenu.com
Sep-15-14:13:47 |  23277 Finished Fake HTTPS: https://share.avvenu.com
Sep-15-14:13:49 |  24867 UDPResponse 66.192.85.140 shoa.bzezxuca.hxmuz.info numRec 0
Sep-15-14:13:50 |  26012 Fetching Fake HTTPS: https://www.cyberbuzz.jp
Sep-15-14:13:52 |  28348 Finished Fake HTTPS: https://www.cyberbuzz.jp
Sep-15-14:13:54 |  30253 Fetching Gdoc HTTPS: https://s3.amazonaws.com/s3c001/2
Sep-15-14:13:57 |  33251 Finished Gdoc HTTPS: https://s3.amazonaws.com/s3c001/2
```

Figure 33: Fetching faux-PGP message and generating chaff HTTPS traffic

Other domains include *user.lolipop.jp*, *www.fotosearch.com*, and many others that appear to be unrelated to UltraReach Inc. It seems extremely dangerous to fetch web pages controlled by an unknown third party with Internet Explorer. Additionally, the timing of these so called *Fake HTTPS* fetches appears to correlate with real fetches of the faux-PGP messages hosted on Amazon S3.

Furthermore the Ultrasurf client appears to parasitically use the Google Web Toolkit service to fetch content as seen in Figure 34.

```
Sep-12-18:18:32 | 42028 Gmobilizer fetching http://googel.de/gwt/n?u=http://114.39.138.221/
    ZWQxNTgwZGVW/
Sep-12-18:18:32 | 42040 Gmobilizer fetching http://google.lv/gwt/n?u=http://rss.od4ihk9.info/
    NDVhYWY0YzL
Sep-12-18:18:32 | 42045 Gmobilizer fetching http://ggoogle.com/gwt/n?u=http://65.49.14.84/
    OWY2ODUwZGN9/1
Sep-12-18:26:37 | 42089 Gmobilizer fetching http://google.co.ck/gwt/n?u=http
    ://114.39.138.221/OGUyMWQ/4O
Sep-12-18:26:37 | 42094 Gmobilizer fetching http://google.bg/gwt/n?u=http://rss.od4ihk9.info/
    YWVjYjg5ZTG
Sep-12-18:26:37 | 42095 Gmobilizer fetching http://ggoogle.com/gwt/n?u=http://65.49.14.56/
    OWY2ODUwZGN9/1
Sep-12-18:36:13 | 42662 Gmobilizer fetching http://google.ie/gwt/n?u=http://114.27.51.197/
    ZGE5NmE5OD/mGH
Sep-12-18:36:13 | 42671 Gmobilizer fetching http://google.pl/gwt/n?u=http://rss.od4ihk9.info/
    OTlmMDBhMT/
Sep-12-18:36:13 | 42675 Gmobilizer fetching http://ggoogle.com/gwt/n?u=http://65.49.14.19/
    OWY2ODUwZGN9/1
```

Figure 34: Fetching faux-PGP message through Google GWT

This use of Google Web Toolkit appears to use a fixed URL size of sixty one characters. Internally the Gmobilizer fetching routines use format strings such as *"Gmobilizer get %s from domain rss.%s.info"* for URL construction. The full set of Google related URLs is listed in Appendix C.

5.14 Client update process

Ultrasurf provides a very minimal in-band upgrade process. The Ultrasurf client connects to the local Ultrasurf proxy on *127.0.0.1:9666* and issues the HTTP GET request as shown in Figure 35.

```
GET http://ultra:80/downloads/ultrasurf/version.txt?
busvntelzlmtiuydkkpvwokmgvmksfsomuvyfjdayqccakkydjtbcmtaioecpisqvtazwtggkguyklffprklbvavuplybaijhyceibxynhebvtag
 HTTP/1.0
```

Figure 35: Version check through the local Ultrasurf proxy

This GET request exposes three important details about the upgrade process—the first is that the proxy is in a different thread from the upgrading process. The second is that each GET request is made for a domain that is not fully qualified. The remote proxy has an alias or a mapping for that special host—this appears to simply be a mapping for ultrasurf.us at this time. The third detail is that the arguments to the request appear to change each time—this appears to be randomly generated data.

```
10.17 9860b1bbf9c34fd466bdd12230c2342c
a2tfVvt1f9xirjcBEWBuHI66QgSmMqqyArnf0M44Keu83EETYj5KkK168X7h1ABR61b61yWEwDLhDBXKcBL0ceVgJhs2bpsnVMdpXapuF8edS

NPHcYEuJaHJnms38BLKyWOmzemnASHWQuqTErzLHRphRLdE
```

Figure 36: Data returned by the Ultrasurf server to the Ultrasurf client

The data returned for the above GET request is shown in Figure 36. The first line is the version number and the corresponding MD5 hash for the corresponding binary. The second and third lines appear to be information used for network discovery. This data is likely the source for the domain name discovery and for other network discovery information. The MD5 hash appears to be used for download verification. In the event that a new version of Ultrasurf is available, the user is prompted to upgrade and if they decline, they will generally be prompted again directly afterwards. If they accept in either case, Ultrasurf will download the file by sending the HTTP GET request seen in Figure 37.

```
GET http://ultra:80/downloads/ultrasurf/u.exe HTTP/1.0
```

Figure 37: The download process to fetch the latest Ultrasurf client software

The file will be saved to the current directory where Ultrasurf is running and the user will be instructed to relaunch that newer copy. It is not actually protected by a digital signature or verified in any meaningful way.

5.15 Client internals

The Ultrasurf client is developed in C++ and is compiled with Microsoft Visual Studio. The source code appears to be managed by either CVS or Subversion.

The Ultrasurf client uses Open and Free Software including Putty [10] and zlib [11]. The use of both Putty and zlib is not disclosed. This use and lack of disclosure is a violation of the licenses. Ultrasurf does not follow the specific licensing requirements for Putty [12] nor the general spirit of the license for zlib [13]. Included copies of zlib and Putty were not up to date with the latest code released by the upstream authors. Given the ease of compliance with the licenses, their decision is bizarre and puzzling.

The Ultrasurf client contains the commonly known RC4, MD4, MD5, SHA1, CRC32 routines for various internal operations. SHA1 is considered reasonable, though deprecated, for use in security related applications. MD5 should no longer be used as it is also deprecated [47]. MD4 and CRC32 are absolutely not safe for any security related purposes. RC4 may be used safely if it is used correctly. It is unclear if RC4 is properly used in Ultrasurf.

The Ultrasurf client lacks any kind of user visible log. Internally the Ultrasurf client log contains information relating to the network status; we found it trivial to extract when desired.

5.16 The Ultrasurf client's local proxy

Upon successful connection to the Ultrasurf network, the Ultrasurf client program opens a local HTTP proxy on *127.0.0.1:9666*. Furthermore, it changes the Windows registry to configure Internet Explorer to use it. It is possible to use other programs with the local HTTP proxy. The local proxy appears to simply forward TCP connections to the current active Ultrasurf server which runs the more complex filtering [4, 5] software. Traffic that enters the local proxy will be emitted by a single Ultrasurf server. The forwarded data is carried in an open TCP connection or a new TCP connection will be opened. This proxy is used by Ultrasurf itself as part of the update process discussed in Section 5.14.

5.17 WJButton

Ultrasurf offers but does not require a plugin, WJButton [48] for use with Mozilla's Firefox web browser to ease integration with the Ultrasurf client. It is originally based on ProxyButton [40]. It provides a way for users to toggle the use of the Ultrasurf proxy in Firefox.

6 Vulnerabilities

The Ultrasurf network and Ultrasurf client are vulnerable to multiple serious issues. The architecture of the Ultrasurf network is an example of privacy by policy and the protection it offers is extremely weak.

6.1 Generic issues

UltraReach as a corporation has extremely questionable data retention practices that include full logging of all user activity [6]. UltraReach is subject to US laws such as National Security Letters [20], subpoena and/or so called *2703 d notice* [14] data production requests.

UltraReach appears to tag their users with third party cookies (Google etc) as well as automatically forcing users to load third party resources. A third party may be subject to the same legal concerns as the UltraReach corporation.

Ultrasurf servers are out of date with regard to commonly used software. Users regularly interface with known exploitable software (Section 6.5) that is multiple years out of date. Publicly available security patches are seemingly ignored. UltraReach server compromise would be a complete break of all of the security properties offered by the Ultrasurf network with the currently deployed architecture. Ultrasurf server compromise would likely allow an attacker to completely compromise specifically targeted clients as well as all connecting clients in an indiscriminate manner.

UltraReach claims in Section 1.1 that a user's IP address will change a million times an hour. This amounts to a new server connection 275 times per second and we find that this is not the observed client behavior.

6.2 WJButton

As explained in Sections 5.11 and 5.17, WJButton falls woefully short of the privacy, security, and anonymity issues covered by similar plugins such as Torbutton [44] and leaves users at risk. It does not block hostile plugins or isolate content. Proxy bypass is avoidable with proper isolation of content and WJButton fails to deliver any meaningful protection at all. User tagging (Section 6.3) is not prevented or addressed by WJButton.

6.3 User Tagging is Deeply Problematic

By default the Ultrasurf client launches an instance of Internet Explorer that visits the Ultrasurf homepage. The URL loaded includes a unique argument at the end of each URL for each visit. Upon visiting this home page, every visitor is tagged with a Google Analytics cookie. When combined with the Google cookie and known server logging [6] information, it appears to individually tag visiting users in a way that creates major privacy concerns. It is possible that the user was previously tagged before downloading and using Ultrasurf. Ultrasurf users who are tagged by Google and other third party cookies are vulnerable to tracking even when Ultrasurf is in use. This tracking continues when Ultrasurf is disabled. Correlation of all web traffic is possible regardless of the browser used because of this method of tagging. This tagging when combined with extensive behavioral logging seems to mitigate any possible claim that Ultrasurf is privacy preserving or anonymity software. Previous visits to websites will be linked to all Ultrasurf browsing as well as all future browsing, with or without the use of Ultrasurf. Ultrasurf does offer an option to isolate cookies but it is not securely implemented, nor is it enabled by default. Their cookie clearing option only covers Internet Explorer and it does not appear to provide protection against forward linking traffic. The use of WJButton (Section 6.2) does not mitigate user tagging issues when using Firefox.

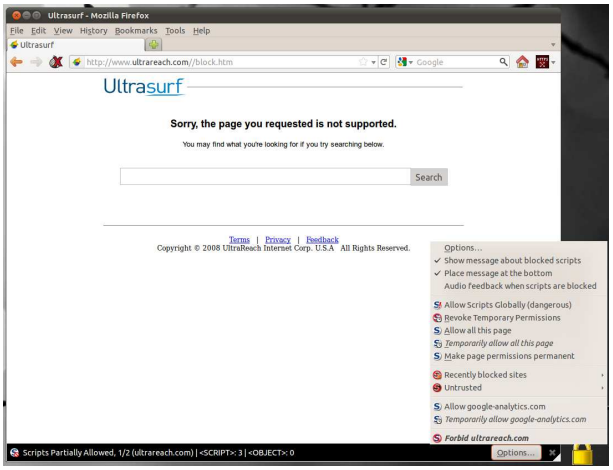

Figure 38: Ultrasurf block page attempting to set Google Analytics cookie

This active tagging indicates that the attack surface for Ultrasurf is incredibly broad. In addition to the Ultrasurf servers and their respective networks, the accounts used for UltraReach's Google Analytics present a very large threat to users. If their Google Analytics account is ever compromised or disclosed, an adversary will have complete logs on almost every Ultrasurf user's behavior. It is extremely likely that such user information would be tied to a wide range of activity on the Internet.

6.4 Controlling Ultrasurf client path selection

The descriptors fetched by Ultrasurf during DNS discovery are automatically used as the first and only hop. An attacker with DNS spoofing capabilities, such as the Great Firewall of China, may successfully return DNS [25] results before remote networks. Such spoofed results would allow an attacker to fully control the Ultrasurf client's path selection process. Barring Man-In-The-Middle protections in the Ultrasurf protocol it may lead to a full compromise of every targeted client and the full network architecture as a result of the Ultrasurf update process as explained in Section 6.6. If an attacker is able to respond with a properly formatted request they will influence the selection of the client's first and only hop. It appears that in some cases Ultrasurf will disregard verification of the remote peer and combined with control of the path a total break of Ultrasurf's protection may be possible.

6.5 Ultrasurf server software

The Ultrasurf server software is out of date with regard to patching of known security vulnerabilities. As an example the Squid proxy used as the core of every connection through the Ultrasurf network was *squid/2.7.STABLE7*, a known exploitable and insecure version [23] of the Squid proxy server.

The web servers detected include lighttpd 1.4.26, Apache httpd 2.2.3 on CentOS, Apache httpd 2.0.63, Apache httpd 1.3 on an unknown distribution of Linux and Microsoft Windows Media Server 9.01.01.5000. Each of these server versions is vulnerable to publically known security issues [15, 19, 23] or it is not the most current version [16] that is widely available.

The DNS servers detected were ISC BIND 9.x and they also appear to be unpatched.

6.6 Subverting the Ultrasurf update process

The Ultrasurf client indirectly downloads updates by fetching a bare executable and it appears to verify that the file fetched is correct by verifying an MD5 hash fetched from the same server. The URL that it uses is not a fully qualified

domain and is likely intercepted by the caching Squid proxy on the Ultrasurf server where a client has connected.

If an Ultrasurf server is compromised, it appears that an attacker would only need to tamper with two values for internally used host names—one is the executable itself `http://ultra:80/downloads/ultrasurf/u.exe` and the other is the text data in the `http://ultra:80/downloads/ultrasurf/version.txt` file.

Figure 39: Ultrasurf's special 'ultra' host and MD5 sums of binaries

6.7 Fingerprinting of traffic

The Ultrasurf client traffic does not appear to be padded, buffered or broken into pieces. This seems to indicate that the traffic would fall to very basic traffic fingerprinting [38]; such an attack would threaten the confidentiality of the traffic and may even allow an attacker to simply block specific content without decryption. Blocking the update process detailed in Section 6.6 seems entirely possible and it would be especially problematic.

6.8 Detection and blocking of access to the Ultrasurf network

Despite claims to the contrary by UltraReach, detection, filtering or blocking Ultrasurf client connections to the Ultrasurf network is possible. While the statements made by UltraReach about Ultrasurf suggest that they have invented a kind of decoy routing protocol such as Telex [49], we find no evidence of such an advanced anti-censorship approach. Rather, we find that it is straight forward to block Ultrasurf. Information on filtering TCP connections to Ultrasurf servers has been previously published [43] although it has generally relied on static bytes in packets. Commercial scale IPS filters for Ultrasurf appear [24] to be available.

Filtering of each of the bootstrapping steps is possible. The static entries in the binary may be extracted and filtered by IP and port number. The DNS bootstrapping phase has a constant byte width for third and fourth level domains even while the second and top level domains may change. Faux-PGP messages may be blocked entirely by simply blocking access to any site that hosts such a message.

Network observation of an Ultrasurf client will allow a censor to simply block access to each host that is accessed. The client will easily serve as an oracle for the censors should they have absolutely no reverse engineering experience. The chaff traffic discussed in Section 5.13 is easily ignored and will not pose a serious problem for censors. If the internal or external encoding of the server descriptor messages changes the client will at some point need to learn about them; thus even without reverse engineering the Ultrasurf client will serve as a network discovery oracle.

6.9 Single hop, single IP

The design of the Ultrasurf system is self-defeating for censorship circumvention and server discovery resistance. Anytime a client visits a web server through the Ultrasurf network, the Ultrasurf server IP address is available to the web server in question. This means that an attacker wishing to block Ultrasurf merely needs to look through log files or run a popular web server for a short amount of time and all of the server IP addresses will be discovered and are thus blockable.

6.10 Data retention

The Ultrasurf network as a whole appears to log connection information for all clients in a privacy-invasive manner [6]. This data is enough to individually identify every user who uses Ultrasurf as directed and to do so after they cease to use Ultrasurf. When combined with the active content (Section 6.2) and active tagging (Section 6.3) issues we find the issue of data retention to be extremely concerning.

6.11 Miscellaneous issues

There are various miscellaneous issues in the Ultrasurf client. It frequently crashes for absolutely no known reason while idle. Perhaps related and also one of the most concerning issues is the use of fixed sized static buffers; some of the static buffers are unsafely used with network supplied data. Another extremely concerning issue is that in some cases a user will believe they are proxied but they are not using a proxy at all. A similar issue exists when the Ultrasurf client itself seems to internally have a race condition and opens URLs that are for internal use while leaking these requests to the public Internet.

Many other best practices about programming and system administration are simply ignored by the Ultrasurf client and the Ultrasurf network.

7 Future Work

We believe that this research lays important ground for future work. Amongst the items we believe need to be further explored, we suggest the following:

- Discovery of all methods of distribution of the top level domains used in bootstrapping
- Decoding of the DNS bootstrapping query and response protocol
- Decoding of the handshake obfuscation process
- Mapping of all Ultrasurf servers
- Extraction of all censorship keywords and URLS
- Tracking binary changes across all versions of Ultrasurf releases

A full third-party client implementation should be a straight forward development task and is merely a simple matter of programming.

8 Conclusion

We have performed the deepest exploration of Ultrasurf to date. We have found the technical realities of Ultrasurf do not match the claims made by UltraReach about the Ultrasurf software. Among the most important finds are as follows:

- We find that Ultrasurf not only leaves traces on the network level, it additionally leaves traces on the system where it is used.
- We find that those traces are enough to leave a uniquely fingerprintable signature for filtering and logging.
- We see that the Ultrasurf network performs censorship as well as actively tagging users with long and short term tracking identifiers.
- We find that Ultrasurf incorporates third party software in violation of their respective licenses.

- We find that they fail to properly patch their servers; this includes the servers that route user traffic.
- We find that they collect, store and share extensive user data and that they share this data with third parties.

Ultrasurf does not provide meaningful anonymity and their security claims are false, misleading or entirely incorrect. It seems reasonable to stress that users who require any kind of security should avoid Ultrasurf. We recommend against the use of Ultrasurf for anonymity, security, privacy or Internet censorship circumvention.

8.1 Disclosure to Ultrasurf

The contents of this report were disclosed in December 2011 to Ultrasurf. They confirmed the contents of this report and explained additional information about Ultrasurf, the design, and the administration of the Ultrasurf network.

Amongst the most alarming admissions from the Ultrasurf team were that log files are indeed being kept, and that they have been disclosed to the US Government without warrants by Ultrasurf. Additionally, it appears that the cryptography in use is even weaker than is described in this paper in extremely alarming ways. They admitted that their protocol has no forward secrecy and that they did not apply an integrity check, such as a MAC or HMAC, when they use RC4 as a stream cipher for client and server communication. Full disclosure of those details is another publication in itself.

After disclosure Ultrasurf took a number of steps, largely superficial, to address the claims in this paper. The claims on the website are now slightly less outrageous. To the best of the author's understanding, they still do not however have forward secrecy in their protocol as of the publication of this paper.

8.2 Ethical liability and delayed disclosure

The nature of Ultrasurf's security and anonymity problems means that delayed disclosure causes ongoing harm. Specifically, its lack of forward secrecy with recorded traffic presents an extraordinary threat to Ultrasurf users. Risk to Ultrasurf users grows over time, and delays in disclosing this report increase the total harm possible for users.

Multiple parties pressured the authors of this paper to delay its release. It is clear that delays are dangerous for Ultrasurf users, and the authors have been against delayed disclosure from the beginning. None of the delaying parties were willing to take on the ethical liability for the dangers exacerbated by the delay in publication of this report.

Disclosure to Ultrasurf in December 2011 confirmed the issues presented in this paper. It was agreed that public disclosure of this report would follow pending a schedule of improvements or evidence that improvements were being made. The authors of this paper have little confidence that such improvements are being made in a substantial manner.

The authors of this paper believe that Ultrasurf must publish technical specifications, a cohesive threat model, publicly viewable source code and submit their design and implementation to a qualified peer review venue.

9 Acknowledgements

This research was only possible thanks to the assistance, guidance and generally positive feedback from Lumineuse, Marsh Ray and a few very talented anonymous researchers who declined to be credited. Furthermore, I'd like to thank Andy Isaacson from Noisebridge, Nick Mathewson, Roger Dingledine and Sebastian Hahn from the Tor Project, Philipp Winter, Jens Kubieziel, Adam Shostock and David Molnar from Microsoft, and Sina Rabbani from Red Team LLC for their feedback on the content of this paper. I'd especially like to thank Nadia Heninger from UCSD for her deep understanding of LaTeX. None of the acknowledged endorse this work or the results but I endorse them. Without their thoughts and help this research would not exist.

References

[1] http://www.oreans.com/themida.php.

[2] http://www.strongbit.com/execryptor.asp.

[3] https://en.wikipedia.org/wiki/Proxy_server.

[4] http://www.squid-cache.org/.

[5] http://ziproxy.sourceforge.net/.

[6] http://www.wired.com/magazine/2010/11/ff_firewallfighters/.

[7] http://www.facts.org.cn/.

[8] http://www.psiphon.ca/.

[9] http://www.jmarshall.com/tools/cgiproxy/.

[10] http://www.chiark.greenend.org.uk/~sgtatham/putty/.

[11] http://zlib.net/.

[12] http://www.chiark.greenend.org.uk/~sgtatham/putty/licence.html.

[13] http://zlib.net/zlib_license.html.

[14] 18 usc 2703 d. https://en.wikipedia.org/wiki/Twitter_subpoena.

[15] Apache httpd 2.2 vulnerabilities. https://httpd.apache.org/security/vulnerabilities_22.html.

[16] lighttpd version 1.4.29 - important changes. http://www.lighttpd.net/2011/7/3/1-4-29.

[17] Malware scan of UltraSurf. http://r.virscan.org/50e6ce64c85e867fbe97da082b985752.

[18] Malware scan of UltraSurf. http://r.virscan.org/report/dd0946d402cddd4de33a4211d52ca5be.html.

[19] ms08-076. https://technet.microsoft.com/en-us/security/bulletin/ms08-076.

[20] Nsl. https://en.wikipedia.org/wiki/National_security_letter.

[21] Psiphon Url Encryption Vulnerability Security Bulletin. http://psiphon.ca/wp-content/uploads/security_1.tx

[22] Randomize the ¨001ïn psiphon URLs. https://bugs.launchpad.net/psiphon/+bug/457377.

[23] Squid security advisories. http://www.squid-cache.org/Advisories/.

[24] Technical tip : How to block ultrasurf. http://kb.fortinet.com/kb/documentLink.do?popup=true&externall

[25] The great dns wall of china, 2007. http://cs.nyu.edu/~pcw216/work/nds/final.pdf.

[26] Is ultrasurf a trojan?, 2009. http://www.how-to-hide-ip.info/2009/01/12/is-ultrasurf-a-trojan/.

[27] 2011. Fetched on November 27th, 2011 http://ultrasurf.us/faq.html.

[28] 2011. Fetched on September 29th, 2011 http://ultrasurf.us/.

[29] 2011. Fetched on September 29th, 2011 http://www.wujieliulan.com/.

[30] 2011. Fetched on September 29th, 2011 http://www.internetfreedom.org/UltraSurf.

[31] U.s. firm acknowledges syria uses its gear to block web, 2011. http://online.wsj.com/article/SB10001424052970

[32] 8e6 Technologies. Ultrasurf whitepaper, 2008. http://www.m86security.com/KB/Attachment33.aspx.

[33] Jacob Appelbaum. https://github.com/ioerror/teatime.

[34] CloneRanger. Cloneranger's comments about ultrasurf client behavior, 2011. http://www.wilderssecurity.com/showthread.php?t=302402.

[35] floss manual. Circumvention tools, 2011. http://en.flossmanuals.net/_booki/bypassing-censorship/bypa

[36] The Berkman Center for Internet & Society. 2007 circumvention landscape report: Methods, uses, and tools, 2009. http://cyber.law.harvard.edu/sites/cyber.law.harvard.edu/files/2007_Circumvention_L

[37] The Berkman Center for Internet & Society. 2010 circumvention tool usage report, 2010. http://cyber.law.harvard.edu/sites/cyber.law.harvard.edu/files/2010_Circumvention_Tool_

[38] Andrew Hintz. Fingerprinting websites using traffic analysis. In Roger Dingledine and Paul Syverson, editors, *Proceedings of Privacy Enhancing Technologies workshop (PET 2002)*. Springer-Verlag, LNCS 2482, April 2002.

[39] Freedom House. Ultrasurf tutorial, 2010. http://www.youtube.com/watch?v=D6B-OkCGr9s.

[40] Oleg Ivanov. http://proxybutton.mozdev.com/.

[41] JanusVM. Janusvm's review of ultrasurf, 2009. http://janusvm.com/Ultrasurf_audit.zip.

[42] Yuma Kurogome. Yuma kurogome's review, 2010. http://webcache.googleusercontent.com/search?q=cache:

[43] Pere Crespo Molina. Filter / firewalling ultrasurf traffic perfectly with iptables or mikrotik, 2010. http://pere.bocairent.net/?p=57.

[44] Mike Perry. https://www.torproject.org/torbutton/design/index.html.

[45] The Free Haven Project. Anonymity bibliography - selected papers in anonymity, Curated selection of publications from 1977 - 2011. The Free Haven Project http://www.freehaven.net/anonbib/date.html.

[46] Bruce Schneier. Snake oil, 1999. http://www.schneier.com/crypto-gram-9902.html#snakeoil.

[47] A. Sotirov, M. Stevens, J. Appelbaum, A. Lenstra, D. Molnar, D.A. Osvik, and B. de Weger. Md5 considered harmful today. In *Announced at the 25th Chaos Communication Congress. URL: http://www. win. tue. nl/hashclash/rogue-ca*, 2008.

[48] UltraReach. http://ultrasurf.us/download/wjbutton.zip.

[49] E. Wustrow, S. Wolchok, I. Goldberg, and J.A. Halderman. Telex: Anticensorship in the network infrastructure. In *proceedings of the 20th USENIX Security Symposium*, 2011.

Appendices

A Imported DLLs

uxtheme.dll, USER32.dll, ADVAPI32.dll, NTDLL.dll, winmm.dll, WININET.dll, ole32.dll, MFC42LOC.DLL, WS2_32.dll, shell32.dll, MSVCRT.dll, KERNEL32.dll, USER32.dll, GDI32.dll, ADVAPI32.dll, SHELL32.dll, shlwapi.dll, COMCTL32.dll, WSOCK32.dll, WINMM.dll, MSVCP60.dll, NETAPI32.dll, imagehlp.dll, etc with KERNEL32.LoadLibraryA() and it uses a few well known Win32 API calls (*_adjust_fdiv, _controlfp, _mb_cur_max, _isctype, _pctype, sprintf, _Strftime, _except_handler3, gethostbyname, RegOpenKeyExW, TerminateProcess, malloc, etc*).

B Hex dump of cached network information files

```
hexdump of Enikbevujkul9l0m:
0000000: 476f 5769 3393 3030 3030 7554 370c 4d3d  GoWi3.0000uT7.M=
0000010: 5c1d ae47 49bb 625e f324 a0bb 7f39 02ec  \..GI.b^.$...9..
0000020: abd5 cc0b 170a 1706 9eba befa            ............

hexdump of Eqklrkiuazud7p7g:
0000000: cc5d fe66 e1c5 fb00 197c 8745 e2e2 ed03  .].f.....|.E....
0000010: 26b3 988b a633 1491 bf53 ad8a 9609 b22c  &....3...S.....,
0000020: 2b41 adf7 5202 2902 9eba befa            +A..R.).....

hexdump of Gmamewmmwymh9d6f:
0000000: 8282 743e 8ba4 2938 8832 b68f 81f4 8114  ..t>..)8.2......
0000010: 66b5 169a 9c30 3698 9a8e ec8f 3608 1194  f....06.....6...
0000020: c8ea 0907 2e47 a54c 689c 9c5a 9aba befa  .....G.Lh..Z....

hexdump of Gpcndcmmksmz7h3h:
0000000: be30 c6bc ff0e 41f5 5df7 49fb 01b3 02e5  .0....A.].I.....
0000010: f312 6fed 0411 e505 ed01 2912 4daf 0816  ..o.......).M...
0000020: c23a e5df 120a d25e a2f4 5d3d 9aba befa  .:.....^..]=....

hexdump of Icmaamqruxrj0t2d:
0000000: 4f34 ca0e f7ba d302 48f6 b7b2 8da6 91d0  O4......H.......
0000010: c2aa a4e2 a1f1 364e 56eb 789d 2fd3 f84b  ......6NV.x./..K
0000020: a6ba befa                                ....

hexdump of Ifobqzdrlmrb8x7x:
0000000: 30f9 6733 ddc4 1b49 b176 b086 0435 61c5  0.g3...I.v...5a.
0000010: d832 54e6 9ca3 0331 2983 60ac c2ed 5d7b  .2T....1).`....]{
0000020: a6ba befa                                ....

hexdump of Yahggswsaysk9w4y:
0000000: 04fc fea6 dfee d668 d0da f561 d663 df9c  .......h...a.c..
0000010: e6a1 5bfc 5b6d 0378 e646 df71 8e68 14b4  ..[.[m.x.F.q.h..
0000020: a6ba befa                                ....

hexdump of Ydjqffjsrssc7a1r:
0000000: b921 66c7 9e71 3657 57d7 cdd3 92fe 79b8  .!f..q6WW.....y.
0000010: 9e62 7597 6c93 ae32 86a7 37fc a482 cb1e  .bu.l..2..7.....
0000020: a6ba befa                                ....
```

Figure 40: Hex dump of files cached in local *utmp* directory

C Google Web Toolkit URL List

The following 332 host names were extracted from the Ultrasurf client and are likely for use with GWT to indirectly fetch content for bootstrapping:
wwwgoogle.com googlecom.com googlebot.com ggoogle.com google.us google.tm google.se google.ru google.ro google.pt google.pl google.no google.nl google.net google.mu google.lv google.lk google.kz google.info google.ie google.fr google.fi google.es google.dk google.de google.com.pr google.com.pl google.com.ph google.com.om google.com.my google.com.mx google.com.jm google.com.br google.com.au google.co.za google.co.uk google.co.nz google.co.jp google.co.in google.co.hu google.co.ck google.ca google.by google.biz google.bg google.be wwwgoogle.com googlecom.com googlebot.com ggoogle.com google.us google.tm google.se google.ru google.ro google.pt google.pl google.no google.nl google.net google.mu google.lv google.lk google.kz google.info google.ie google.fr google.fi google.es google.dk google.de google.com.pr google.com.pl google.com.ph google.com.om google.com.my google.com.mx google.com.jm google.com.br google.com.au google.co.za google.co.uk google.co.nz google.co.jp google.co.in google.co.hu google.co.ck google.ca google.by google.biz google.bg google.be adwords.google.cn adwords.google.co.jp adwords.google.com checkout.google.com google.com groups.google.com mail.google.com services.google.com upload.video.google.com www.google.com wwwgoogle.com googlecom.com googlebot.com ggoogle.com google.us google.tm google.se google.ru google.ro google.pt google.pl google.no google.nl google.net google.mu google.lv google.lk google.kz google.info google.ie google.fr google.fi google.es google.dk google.de google.com.pr google.com.pl google.com.ph google.com.om google.com.my google.com.mx google.com.jm google.com.br google.com.au google.co.za google.co.uk google.co.nz google.co.jp google.co.in google.co.hu google.co.ck google.ca google.by google.biz google.bg google.be wwwgoogle.com googlecom.com googlebot.com ggoogle.com google.us google.tm google.se google.ru google.ro google.pt google.pl google.no google.nl google.net google.mu google.lv google.lk google.kz google.info google.ie google.fr google.fi google.es google.dk google.de google.com.pr google.com.pl google.com.ph google.com.om google.com.my google.com.mx google.com.jm google.com.br google.com.au google.co.za google.co.uk google.co.nz google.co.jp google.co.in google.co.hu google.co.ck google.ca google.by google.biz google.bg google.be wwwgoogle.com googlecom.com googlebot.com ggoogle.com google.us google.tm google.se google.ru google.ro google.pt google.pl google.no google.nl google.net google.mu google.lv google.lk google.kz google.info google.ie google.fr google.fi google.es google.dk google.de google.com.pr google.com.pl google.com.ph google.com.om google.com.my google.com.mx google.com.jm google.com.br google.com.au google.co.za google.co.uk google.co.nz google.co.jp google.co.in google.co.hu google.co.ck google.ca google.by google.biz google.bg google.be wwwgoogle.com googlecom.com googlebot.com ggoogle.com google.us google.tm google.se google.ru google.ro google.pt google.pl google.no google.nl google.net google.mu google.lv google.lk google.kz google.info google.ie google.fr google.fi google.es google.dk google.de google.com.pr google.com.pl google.com.ph google.com.om google.com.my google.com.mx google.com.jm google.com.br google.com.au google.co.za google.co.uk google.co.nz google.co.jp google.co.in google.co.hu google.co.ck google.ca google.by google.biz google.bg google.be wwwgoogle.com googlecom.com googlebot.com ggoogle.com google.us google.tm google.se google.ru google.ro google.pt google.pl google.no google.nl google.net google.mu google.lv google.lk google.kz google.info google.ie google.fr google.fi google.es google.dk google.de google.com.pr google.com.pl google.com.ph google.com.om google.com.my google.com.mx google.com.jm google.com.br google.com.au google.co.za google.co.uk google.co.nz google.co.jp google.co.in google.co.hu google.co.ck google.ca google.by google.biz google.bg google.be

D Hashes of collected Ultrasurf binaries

The following is a list of information about Ultrasurf executables found in the wild. Such a list is sometimes called an archeology study; we list the file name, the SHA1 hash of the file and the file size to assist in future Ultrasurf binary archeology studies:

u1000.exe sha1sum: 6e8a404b264cff6a20986af8cfd653c19e72ddf8 file size: 544K
u1001.exe sha1sum: 0dd92b15f98ecff2eb8a302508c8d0500a2c1ebb file size: 544K

u1002.exe sha1sum: 7de60092dc427372264110668a8df92f180e8c62 file size: 760K
u1003.exe sha1sum: 7a94738220c097981b419b7d0c72f66aaddef27e file size: 924K
u1004.exe sha1sum: 62f1d6d584bba8a96db190ff7d7f3e807ee63463 file size: 1.2M
u1005.exe sha1sum: 6c3dc8fd61dcc1b33a70b1a1190957907851c027 file size: 1.2M
u1006.exe sha1sum: 4e864b277fe350a30e25fdd703038ea0f17a3f2a file size: 1.3M
u1007.exe sha1sum: 859fddd98512620c2b086ac73f240566cd3617ea file size: 964K
u1008.exe sha1sum: 3efa10b5724887b0dee11b7f9948232517050d6d file size: 1.1M
u1009.exe sha1sum: bea92123bc4e62271d78d397a4c002d2962dc8d7 file size: 1.1M
u1016.exe sha1sum: 31706fa9431f848ceea9b21a25ccf7850198ee24 file size: 1.1M
u1017.exe sha1sum: ad70593e95b53075290c5ecbf411dac8dba3c4b5 file size: 1.1M
u1102.exe sha1sum: d8671cf1ebf2afeb6fda9228aa7789b8e0953d7f file size: 1.3M
u1103.exe sha1sum: 08a234aa86036fcd1a208994b88668ee5ac0b851 file size: 1.2M
u1104.exe sha1sum: 7f7183d5b5acf94a61b4e0dfe82b45a5ace838bd file size: 1.4M
u60.exe sha1sum: 6db58e3bd0b964a65a65bb5342abe67bbe25961c file size: 112K
U85.exe sha1sum: f87b98c37359bb077574dab9fa396dd690d19c91 file size: 96K
U8.8.exe sha1sum: 5668abd023092addb262e105bca63eaa85d6133b file size: 172K
u95.exe sha1sum: dd1fccb97d90f4aa00a2bed174dba1e4d9e87df4 file size: 456K
u96.exe sha1sum: 7b6d5e2aad897b2dfbc5d596202f93cae6b87e67 file size: 428K
u97.exe sha1sum: c2cbc2c68a9d2ae6fa4c0dfbe5fd7b8e92c25112 file size: 420K
u98.exe sha1sum: 281997156a19852efafd06b5ab97c21d5c90d111 file size: 424K
u991.exe sha1sum: 45107d3d37ee57f8ca5b46e8440e80515e206017 file size: 416K
u992.exe sha1sum: 017c1f5cb308953c40568953a19d1d8cc1bfe5a3 file size: 424K
u993.exe sha1sum: 8dcc53d0a6b95430c7cd07ab1af54b040a0edfd2 file size: 424K
u994.exe sha1sum: 3bc9c76150c9c84b14a218ef07a870783e7afb9d file size: 424K
u995.exe sha1sum: 79f0b75482a086c831adff7a33df19c912ef4baa file size: 428K
u996.exe sha1sum: 11186f9c8f724218e13ad02a711870b6d3801b36 file size: 500K
u997.exe sha1sum: 584c78870b7150fc4a0dd76ca0047ff84b4851d1 file size: 500K
u998.exe sha1sum: 6de82d41432fc04844bf642b558404ca3f61bbee file size: 492K
u999.exe sha1sum: d0bac72aff829455fb02c81be1f15b0d5d2c7f94 file size: 496K
u99.exe sha1sum: 5e3ca21305d3656da463d501dece0dfa37ae767c file size: 420K
UltraSurf6.0.exe sha1sum: 6db58e3bd0b964a65a65bb5342abe67bbe25961c file size: 112K
UltraSurf62.exe sha1sum: 260abfb7c703c75228145323a1b3322beca0bafe file size: 104K
Ultrasurf8.0.exe sha1sum: ba088b3f66944bb8f47c9e23ea46acf59a4cb029 file size: 92K
Ultrasurf8.7.exe sha1sum: 86b7703aaf614a8a0276552173c69c7ff61479ec file size: 104K
UltraSurf8.8.exe sha1sum: 39f66f55036686fccf8090121a1b5367ed29a6f4 file size: 176K