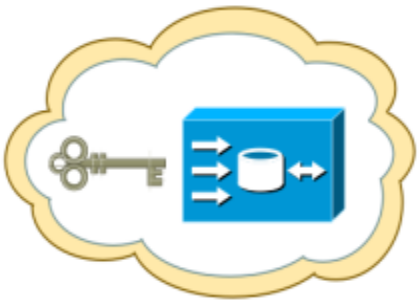
PROXY AND HTTP AUTHENTICATION AND AUTHORIZATION NEEDS A REBOOT



As the IT industry continues to transform itself more and more towards a cloud based model, the walls of our organizations separating inside from outside or trusted from untrusted are slowly disappearing. With the desire to save money by leveraging cloud based solutions and the demand of business users to gain mobile access to their applications, it is presenting a big breakdown in the traditional security model. Even the existing approaches are a disjointed collection of distributed capabilities within individual applications and a wide variety of endpoint flavors. This leads to non-uniform user experiences across platforms and a complex and costly operational support problem for IT organizations.

Generally, anyone who works in Information Security will agree, to a large extent, that the more distributed security controls are and the more entry and exit points to and from untrusted environments, the less effectively they are managed and maintained. Thus, this approach diminishes the overall security posture. It is similar to the principals relative to physical security. There is a reason why secure companies have only one or maybe two entrances into a facility, depending on the size. I see this distributed security paradigm as a big problem for the IT industry in that companies are really interested in

taking advantage of cloud services, yet the security models and capabilities are quickly being outdated and becoming increasingly difficult to implement and maintain because they are so distributed. The majority of risk management sensitive companies will face huge challenges trying to move their internal applications and data to the cloud, due to the existing band-aid security models that can not address the whole threat landscape or do it in a native fashion. These deficiencies will challenge organizations ability to scale and provide the ubiquitous access they are looking for.

Without new security models, most organizations will latch onto the traditional model of network containment by location. Source IP filtering. This model is beyond outdated in today's mobile climate. The problem companies have today is how to know if the end point connecting to their apps and data can be trusted or are "clean" or not. Enter the capabilities of NAC or Network Admission Control. NAC is a great idea but, again, it generally only addresses the problem based on a physical location model. You have to physically or logically connect to a network or VPN service to be subject to controls such as 802.1x. This still does not

answer the mobile use case where source location can be anywhere. Sure you can force all your endpoints to come back through a VPN gateway to force inspection, but this diminishes the overall user experience because it is not native, it becomes a 2 step process to gain access, and is generally not seamless and incurs a fair amount of overhead and performance loss. If I could point to a single challenge that is killing the ability of enterprises to be more mobile with their data and applications and their end users to display widespread adoption with these channels it is the lack of the native user experience. No one really wants to sit on their smartphone or tablet and fire up the VPN client and find their 2 factor auth token and enter that all just to get connected, so they can launch a quick application. It is a terrible user experience. We want to launch the app(s) and connect once and stay transparently connected through IP and network changes.

So what should we be doing to improve these security models across the industry? We need native and integrated capabilities in the predominant application protocol, HTTP, that can open up and level the playing field for application control and access management. We need to have NAC capabilities natively integrated at the application layer. With an open framework, any vendor can plug into this capability through the browser (client), a proxy, a webserver, or any combination thereof.

In the Enterprise, the need for application layer visibility and control continues to grow on a constant basis as the boundaries between trusted and untrusted endpoints continues to be blurred. A common approach is to leverage a forward proxy platform to deliver this capability. Alternatively, to a lesser degree, a reverse proxy might be used in front of specific applications or a suite of applications.

Anyone who has worked with proxies knows the challenges and limitations with implementing almost any method of client authentication. The built-in methods such as Basic Auth and Digest in the HTTP specification are really intended for the endpoint content servers themselves and not the proxy in the middle. Additionally, there is no native capability to provide for authorization, aside from leveraging the supplied credentials to query for further group or attribute information. That type of authorization is only about the user and provides no additional value with respect to the source device or platform. Regardless, proxies continue to rely on an outdated and insecure model to achieve authentication using the 407 challenge and response. Essentially it is easy to deploy, it just works in the browser, and the user experience is well known. The problem with these methods is that Basic Auth is quite insecure in that it only uses a Base64 encoding scheme on the credentials when they are transmitted. The Digest method can’t work with a proxy because the credentials are transmitted using a one-way hash that the proxy has no method of intercepting.

There are other methods of performing client authentication to a proxy like virtual auth redirect or client certificates but these methods either don’t scale well to high volume usage or are difficult to manage, especially with a large proxy environment where users need to remain evenly distributed and persistence is generally viewed as a negative behavior. Additionally, with the virtual auth redirect method, the client must manually go through that same credential exchange process any time they migrate between proxies in a cloud service. As the users move across proxies and data centers for different content requests, there would be a constant frontend churn for authentication processes which is a poor user experience. Proxies can be very high volume environments. Little problems that require multiple turns back and forth get amplified enormously in terms of load and resource consumption. Basically, my point is that the state of authentication and authorization on proxies is sorely lacking particularly when it comes to the ability to manage it within a

large scale deployment and to provide the additional authorization capabilities needed as mobility and cloud deployments become more pervasive.

I believe the technology needs a better way to handle this capability. That is why I am proposing a new open standard framework for proxy authentication and authorization to overcome this challenge. The method should leverage the well-known browser user experience, needs to be secure between the client and the proxy, extensible so that it can work across any proxy client and support a wide variety of authentication schemes, and seamless so that session authentication can be carried across proxies and content servers transparent to the end user, even with a front end load balancing solution.

In order to achieve security between the client and the proxy, there needs to be a way to seamlessly establish a secure communication channel from the client to the proxy upon the authentication challenge. SSL makes the most sense, since it is quite ubiquitous with HTTP and the majority of clients and proxies already support this methodology and include the necessary libraries. The key is that once the credentials have been securely transmitted, authenticated, and authorized that the proxy and the client must be able to switch back to a potentially insecure communication channel, yet still retain a credential surrogate for validation or future authorization transactions. The easy part is the initial secure communication transaction and credential exchange. The challenge is implementing the secure credential surrogate to be used on insecure channels and to implement the authorization framework.

Following is an explanation of how my proposed secure proxy authentication method would work:

- A client makes an HTTP request through a proxy. The proxy determines the client is not authenticated or authorized but is required to be by policy for the particular request.
- The proxy sends an HTTP 418 secure proxy auth challenge response that includes the proxy IP and or fully qualified hostname, service port, a defined list of supported authentication methods, and, if it is manual credential auth challenge, an optional text message to include in the authentication prompt generated by the client.
- The client application will select an offered authentication method based on a configured priority or by default.
- If the auth method selected is Basic, the browser will then popup a secured authentication prompt when it receives an HTTP 418 response, including the optional text message provided by the proxy. Go to Step 6.
- If the authentication method selected is Client Certificate, the client will immediately establish an SSL connection to the proxy using the name or IP and service port, where it will then be presented a Client Certificate Challenge. Go to Step 7.
- Once the end user has submitted the credentials locally within the client and clicks OK, the client subsequently establishes a direct SSL connection to the offered IP of the proxy or looked up IP of the proxy and service port.
- Once the SSL session is established to the proxy, the client locally caches and then transmits the submitted credentials to the proxy through the SSL connection including the selected Auth method and the Referrer URL. When the client submits the credentials, it additionally includes a standard list

of uniquely identifying information the proxy or content server can use along with a start of session timestamp to seed the generation of the secure token. This allows the proxy to revalidate the client to prevent malicious token reuse in the event that the source IP address changes.

- The proxy then authenticates the user on the backend, using the defined authentication realm for the selected auth method. For consistency of security policy, the proxy should require the use of an authentication realm that uses secure communication i.e. LDAPs.
- If authentication is successful, a positive response or HTTP 200 OK is returned to the client with a unique Secure-Proxy-Auth session token generated by the proxy. The generated token should be a hash of the unique client information that was provided by the client plus a session timestamp value which can be revalidated to prevent unauthorized token reuse.
- The proxy caches the authenticated session based upon the originally requested URL and the newly generated session token and then tears down the SSL session with the client.
- The client then reuses the initial TCP connection and submits a new HTTP request for the same URL to the original proxy address including a "Secure-Proxy-Auth:" header with the proxy name and the associated token as a key pair. The proxy can then identify the unique user and authorize the user session based on user attributes, if required by policy, based on the provided user credentials associated to the generated session token.
- After a client is authenticated, there may be additional client authorization checks that need to occur before permitting access. If so, instead of returning an HTTP 200 OK response, the proxy would return an HTTP 419 Authorization required response.
- Once the session has been authorized, the proxy may cache the authorization information and the user token for further validation.
- If authentication is unsuccessful, the proxy returns a subsequent HTTP 418 challenge to prompt the end user to enter credentials again. The client can reuse the existing SSL connection to encrypt the credentials to transmit to the proxy. The SSL session should not be torn down, unless an HTTP 200 OK or 403 Forbidden response occurs. The proxy may proactively tear the session down.
- After a default of 3 or a configured number of failed authentication attempts, the proxy may respond with a 403 Forbidden response ending the challenge loop and allowing the SSL session to be torn down to conserve resources.
- Once the initial authentication process has completed, the client has a secure token it can offer to a particular named proxy on all subsequent requests.

Following is an explanation of how my proposed secure proxy authorization method would work (this is still a work in progress):

- When the initial authentication phase concludes, at Step 9 above, if further client authorization is required by policy, an HTTP 419 Authorization Required response would be generated, including a header with an application URL for the subsequent client scan and inspection.
- The client will request the provided application URL through the existing SSL connection and the proxy will mediate the connection. The proxy can act locally as the posture assessment platform and

perform all tasks natively or rewrite the request to any external posture assessment platform or broker service to initiate the scan, including whatever the defined requirements are in policy for the client posture.

- At this point, the designated service for posture assessment will perform its initial client detection and checks against the client to ensure the necessary agent is installed and, if not, delivering the necessary component to execute the host scan. If the initial agent check or install fails, Authorization would return a 403 Forbidden to the client.
- Once the client posture assessment is completed a response is returned to the proxy with either an HTTP 200 OK success or an HTTP 403 Forbidden.
- In the event of a successful authorization, the proxy would cache the Authorization as part of the user session along with an associated policy attribute as defined by the proxy. In the event that further client authorization is required on subsequent transactions, the existing authorization could be evaluated for the matching policy attribute. If the subsequent authorization policy is the same and the client previously authorized successfully within a predefined maximum time interval, it would not require a complete authorization process to be executed again.
- Once the client is authorized with a 200 OK response, the client would, once again, reuse the initial TCP connection to the proxy to request the original URL, including the secure-proxy-auth header with the unique session token.

Source : https://rolande.wordpress.com/2013/03/04/proxy-and-http-authentication-and-authorization-needs-a-reboot/