Transitioning GlideinWMS, a multi domain distributed workload manager, from GSI proxies to tokens and other granular credentials

Marco Mambelli[1,*], *Bruno* Coimbra[1], and *Dennis* Box[1]

[1]Fermilab, PO Box 500, MS 120, Batavia, IL 60510

Abstract. GlideinWMS is a distributed workload manager that has been used in production for many years to provision resources for experiments like CERN's CMS, many Neutrino experiments, and the OSG. Its security model was based mainly on GSI (Grid Security Infrastructure), using X.509 certificate proxies and VOMS (Virtual Organization Membership Service) extensions. Even when other credentials, like SSH keys, were used to authenticate with resources, proxies were also added all the time, to establish the identity of the requestor and the associated memberships or privileges. This single credential was used for everything and was, often implicitly, forwarded wherever needed. The addition of identity and access tokens and the phase-out of GSI forced us to reconsider the security model of GlideinWMS, to handle multiple credentials which can differ in type, technology, and functionality. Both identity tokens and access tokens are supported. GSI proxies even if no more mandatory, are still used, together with various JWT (JSON Web Token) based tokens and other certificates. The functionality of the credentials, defined by issuer, audience, and scope, also differ: a credential can allow access to a computing resource, or can protect the GlideinWMS framework from tampering, or can grant read or write access to storage, can provide an identity for accounting or auditing, or can provide a combination of any the formers. Furthermore, the tools in use do not include automatic forwarding and renewal of the new credentials so credential lifetime and renewal requirements became part of the discussion as well. In this paper, we will present how GlideinWMS was able to change its design and code to respond to all these changes.

1 Introduction

GlideinWMS has been an early adopter of JWT-based token authentication. This has been quite a ride with many considerations and lessons learned that could benefit other software going through a similar transition.

The next section will introduce the GlideinWMS system and the following will introduce the security model. Then two sections will present how credentials were handled with X.509 and the model discussion and redesign ushered in by tokens. Finally, this paper will conclude with the lessons learned during the implementation of token support in GlideinWMS and the rationale of the credential handling refactoring currently in progress.

*e-mail: marcom@fnal.gov

2 The Glidein Workload Management System

GlideinWMS [1, 2] is a pilot and pressure-based Workload Management System (WMS) provisioning computing resources in a distributed environment. Its users can request one or more customized elastic HTCondor Software Suite (HTCSS)[3] clusters, User Pools, in green in figure 1, where the users run their computations. To do so GlideinWMS sends to a variety of computing resources as shown in figure 1 Glideins, also called pilot jobs to distinguish them from the scientific computations, the user jobs. It has been and is used at scale in production by many collaborations, including the Compact Muon Solenoid (CMS) experiment, many Fermilab experiments, and the OSG for more than 10 years. The Virtual Organization, VO, is the computing model abstraction of these collaborations, so we'll use the terms interchangeably. Most scientists will not use directly GlideinWMS or the clusters it provides, they will interact instead with the various tools or portals like CRAB, JobSub, or OSG-Connect, provided by the scientific collaborations.

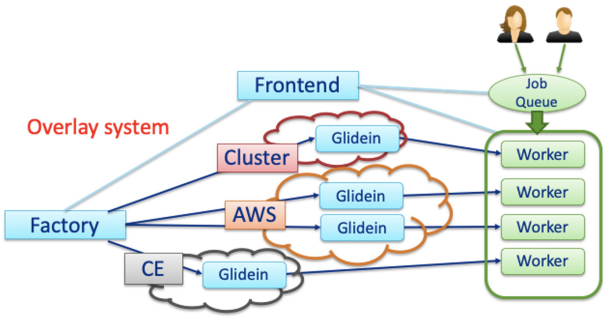

Figure 1. GlideinWMS system. GlideinWMS components are in blue, the User Pool is in Green, and the computing resources are in other colors.

The key component is the Glidein, the pilot job. It is a program, sent to many resources that match the preliminary requirements of the user jobs, to test and set up each computing resource to run the user jobs. It can auto-detect and report node resources like CPU cores, memory, disk, and GPUs, can install common resources like a container runtime or a distributed file system, and provides monitoring and audit information. It also stores and uses pilot credentials and it safely receives and stores user job credentials. It finally joins the User Pool to run one or more user jobs, in parallel and in sequence depending on the needs and availability.

The Factory and clients like the VO Frontend or HEPCloud's Decision Engine[4] complete the GlideinWMS system. For clarity, in this paper we'll consider a system with one Frontend, one Factory, and their Glideins. Actual deployments may include multiple clients, differing in how they calculate the requests for the Factory, and multiple Factories, providing a redundant distributed system.

The Factory is in charge of submitting Glideins to the different Compute Entrypoints, CEs. It knows how to reach each computing resource, which VOs are supposedly supported, which protocols and authentication methods are supported, and if there are throttling requirements. It submits Glideins to the Compute Entrypoints maintaining on each one the pressure, i.e. number of queued and running Glideins, requested by its clients. The Factory monitors the Glideins and it hosts a secure mailbox used to exchange requests and status messages with the clients. It also caches the credentials necessary to submit Glideins or to be forwarded to the Glideins.

The Frontend and other clients are aware of the users' requests and the running and queued Glideins that can be used for those requests, they receive resource status information from the Factory, and they use heuristics to update the requests to the Factories so that all the user jobs can run promptly, all limits and policies are respected, and no resources are wasted. The Frontend is generally operated by the VOs or on their behalf and is the custodian of most credentials used in the GlideinWMS system: the ones to authenticate with the Factory, the ones to submit Glideins to CEs, the ones to join the Users Pool, and the ones Glieins may use to access VO-level services like monitoring, databases or storage volumes.

3 Security domains and federated trust

The framework is distributed and allows many to many connections: there can be multiple Frontends, each providing clusters for multiple collaborations and talking to one or more Factories, dedicated or shared by multiple Frontends. Each Factory talks to many resources and the sets of resources available at each Factory generally overlap.

Figure 2 shows the three types of security domains involved:

1. The VO, managing the Access Points to submit user jobs and frequently owning the Frontend or at least the credentials used in there.

2. Factory operation, frequently handled by consortia, like OSG, since a single Factory can handle several Frontends and hundreds of resources. and concentrating in the hands of a few experts allows economies of scale.

3. The resource owners, universities, laboratories, or commercial operators managing the computing resources, like clusters or clouds, and granting access to run the VO jobs.

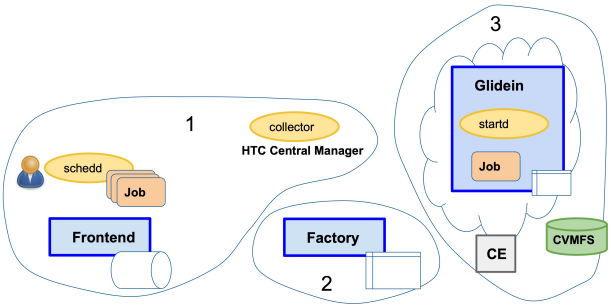

Figure 2. Security domains in the GlideinWMS system.

Each VO provides user credentials, certifies whether a user belongs to it, states the capabilities of each user, and negotiates resource access and priorities with the resource owners. Factory operators are trusted third-party operators. Each resource owner manages the computing and storage resources and enforces arbitrary systems to authenticate the users and allow access. These are normally universities, National Laboratories, or commercial entities like AWS or GCE, which negotiated collective policies for some VOs. There is a transitory trust: VOs certify their members, and resource providers allow access to VOs. This way users can run on the resources. In X.509 VOs maintain a membership server, VOMS [5], and sign proxy extensions. Token providers use information from institutional databases and issue tokens accepted by the service providers.

4 The GSI authentication model

In the X.509 model hosts and users are identified by certificates with appended signed VO membership extensions. Sometimes proxies are used instead of the certificates to improve security by limiting time or scope. Each host, client or server, is using its certificate to prove its identity. Each user is forwarding her/his certificate, or a proxy, with all her/his computing or storage requests. Two facts make X.509 relatively easy for service implementers: there are few credentials involved and it has been around for over 20 years, so it has been integrated into most systems and has many tools. The X.509 proxies are frequently forwarded and renewed automatically making them de facto "omnipresent and all-capable".

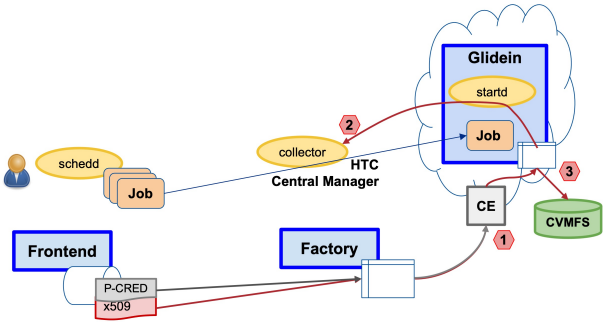

Figure 3. X.509 authentication in GlideinWMS. X.509 credentials, in red, move from the Frontend storage (cylinder) to the caches (tables) and are used multiple times (hexagons).

As visible in figure 3, certificates or proxies, in red, are placed in the Frontend and forwarded and cached in the Factory. They are used in three different ways: the Factory uses them most of the time to have access to the CE (1) and forwards them to the Glidein where they allow the Glidein to join the User Pool (2) and to access VO services like accounting, storage volumes, databases, and monitoring (3). Also when a different credential is used to access a CE (1), like the SSH keys used for some High Performance Computing (HPC) resources, the proxy is still traveling along and forwarded to the Glidein for all other functions (2,3).

In the next section, we'll see how this model changes when generalized and when tokens are used.

5 The granular model of JWT

JSON Web Token (JWT) is a proposed Internet standard[6] for creating data with optional signature and/or optional encryption whose payload holds JSON that asserts some number of claims. The tokens are signed either using a private secret or a public/private key. JWTs are widely adopted in the industry

The reasons behind the transition from X.509 to tokens are discussed at length in Dack et al. [7], here we'll focus on the consequences for the security model and the service implementations.

Credentials have a purpose, a type, and a time and space granularity. X.509 proxies were generally multi-purpose and long-lived, tokens are generally short-lived, very granular, and have a single purpose. As shown in figure 4, in the GlideinWMS system we identified a few functions that can be fulfilled by different credentials:

- the cluster credentials, C-CRED in yellow, which allow a Glidein to join a User Pool
- the pilot credentials, P-CRED in gray, used at the CE to access the computing resources
- multiple service credentials, S-CRED in green, used by the Glidein to access VO services like authenticated monitoring services, storage volumes, or databases, accessible to all jobs running on that Glidein and belonging to the same VO.
- User jobs can bring multiple job credentials, J-CRED in orange, e.g. to read input files or write output files.
- special purpose credentials are issued and verified by the same service, like the CE credential CE-CRED in dark gray, provided by the CE and used by the Glidein to record monitoring and audit information

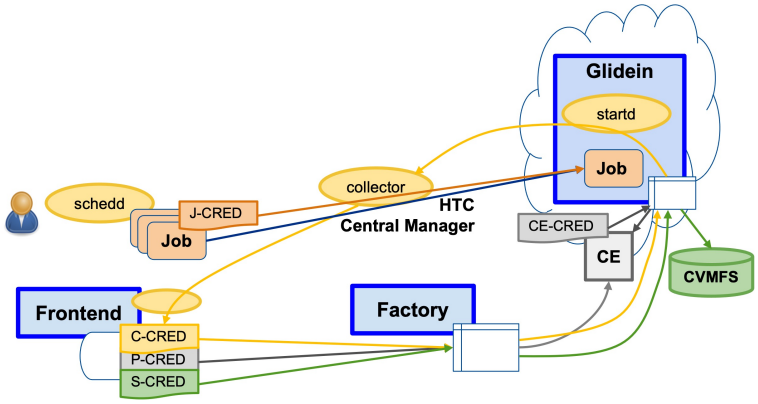

Figure 4. Token authentication in GlideinWMS. Many different single-purpose credentials (document icons of different colors) are stored, forwarded, and used throughout the system.

GlideinWMS was already supporting multiple credential types, like X.509, and SSH keys, but they were always companions of the main X.509 proxy. Now is important to declare the purpose and to define whether credentials are forwarded, used together, or if one is a fall-back alternative. This complicates the specification of available credentials, in the Frontend, the specification of required credentials, in the Factory, and the matching to understand if a group of User jobs has all the required credentials to run on a given resource.

The increased spatial granularity brought new complexity as well. To mitigate against compromised resources or Man-In-The-Middle attacks, each CE is sent a different credential. Since there are too many resources in a Factory to pre-generate short-lived fresh tokens for all of them and since a Frontend may know which resources are available only at run-time when a match is attempted, the tokens must be generated on the fly after the matches.

Mechanisms need to be in place to forward and renew credentials. These are available and implicit for X.509 but are missing for tokens. And the increased granularity of tokens both in the space and time domains makes this even more pressing. A Glidein may be queued at computing resources for several hours, up to a few weeks on HPC resources, and then it will run for many hours generally up to a handful of days. During this whole time, it needs valid credentials to connect to the User Pool and access VO storage resources. A single proxy could outlast the duration necessary for a Glidein to complete even in the worst-case scenario. Not so for JWT. A new failure mode became visible when a production User Pool was restarted: HTCSS lost all the sessions' information. The system was normally resilient to these restarts.

When the Glideins tried to establish a new session with the restarted User Pool they failed because their tokens were expired and this caused all the Glideins and the jobs they were running to die. A token refresh mechanism is the obvious solution but alternatives are being investigated, like modifying the HTCSS main server to checkpoint session information and accept the old one right after a restart.

6 Adding token support to GlideinWMS

The support of tokens in GlideinWMS has been a long process with some notable lessons learned. We started discussing tokens in the Spring of 2019 and they became a high priority in August 2019 when the OSG technical lead declared the intention to drop X.509 support by early 2020. The project dedicated a developer for the effort and the plan was to use token-auth, the HTCSS implementation of JWT, now IDTOKENs, to authenticate Glideins with the User Pool and to authenticate Frontends with Factories, and to use resources supporting INDIGO IAM [8] and SciToken [9]. By October we had a prototype authenticating only with tokens: Glideins are very flexible and we added a token-auth token to the payload to authenticate with the user pool and changed the Factory to use INDIGO tokens to authenticate with a test CE. X.509 proxies were still in the configuration and forwarded through the system, only they were not used. OSG plans changed and so did some of the underlying technologies, e.g. HTCSS moved to IDTOKENS. Took about three years to have a production-ready GlideinWMS supporting tokens, partly due to COVID-19 disruptions, mostly due to technical complications. In March 2020 GlideinWMS v3.7 was using token-auth for the Glidein if all the underlying HTCondor versions were supporting it. In November 2020 v3.7.1 added token support for submission to resources, added token authentication between Frontend and Factories, and adapted to the changes in HTCondor, now using IDTOKENS. 2021 was used to remove X.509 requirements (until now tokens were sent in addition to X.509 proxies), to fix bugs highlighted by wider adoption, and to implement transition mechanisms. The initial OSG plan for a hard transition gave place to a more gradual one and the need for fall-back mechanisms if some resources or some components were not supporting tokens. The following releases added more granularity generating different User Pool tokens for each resource and with versions 3.7.5 (Python 2) and 3.9.5 (Python 3) tokens started being used in production by OSG and CMS. It is only at the end of 2022 with v3.10.1, full support of credential generators, token support for Glideins running on Cloud and HPC, and better handling of fall-back to X.509, that we consider the transition complete. The next step, still in progress, is a refactoring and hardening of all the GlideinWMS code handling credentials.

Being an early adopter, exposed GlideinWMS to surprises and changes in other systems and requirements. We started working with a custom build of HTCSS because token support was not in any release, then we used many release series, from v8.9.x to v9.x, v10.x, and now v.23.x series, encountering undocumented features and many changes in the tokens and security model. Initially, we were expecting a hard transition but then became evident that a soft transition, with fall-backs, multiple credentials at the same time, and features to reduce the Factory operators' load and be resilient to misconfigured CEs, was important. Understanding better the capabilities of tokens added new requirements, like generating different tokens for each CE, to increase the system security in case of a compromised CE. X.509 credentials had many tools, to verify them, to auto-renew them, to forward and renew them. Many tools for tokens have been added, but token renewal remains a problem, exacerbated by its recommended short lifespan. Once we started to remove X.509 proxies, we kept discovering components of the ecosystem GlideinWMS is part of that relied on them: the CE monitoring, audit logs, and OSG accounting to name a few of the late ones. Last but not least, X.509 is very ingrained in the GlideinWMS code: most functions or variables are designed

for X.509 Proxies and are also called *Proxy...*; many checks interspersed in all code make sure that proxies are present, configured correctly, and forwarded all the way to the Glidein. This had to be changed and created for tokens.

Many problems encountered have been solved thanks to collaborations with many different groups. GlideinWMS had already a strong collaboration with OSG, operating GlideinWMS servers in production and supporting the software release distribution, and with the HTCSS developers in Madison, WI, since GlideinWMS relies heavily on HTCondor for the User Pools and mechanisms to access the computing resources. Having a direct line with the HTCondor team was instrumental in getting quick changes, suggestions and help with undocumented features. And the OSG VO and Factory operators helped ironing out the initial production deployment. SciToken and INDIGO IAM provided the initial training and tools to create and use tokens. And the participation to the weekly call of the WLCG Security Group helped GlideinWMS to be informed about token evolution and gave the opportunity to provide feedback from practical use cases.

Being a trailblazer, GlideinWMS contributed to pushing forward the transition to tokens described in Dack et al.[7]. It tested new technologies, discovered shortcomings evident only after running actual systems without X.509, and helped resolve them. The GlideinWMS team hand-held the first VOs transitioning to tokens, helping with solutions involving the whole ecosystem. GlideinWMS aggressive schedule may be one of the reasons why WLCG and OSG computing are fully transitioned to tokens, ahead of storage.

7 Code refactoring

Implementing support for JWT in GlideinWMS highlighted a few problems with the way our software handled credentials. Before JWTs, we already supported several credential types, such as X.509 certificates (and proxies), RSA keys, and the traditional username and password. Adopting JWTs required us to add two new credential types to the list: SciToken, covering both SciToken and INDIGO IAM profiles, and IDTOKEN, both variants of the JWT standard.

The code handling the credentials was mostly not Object-Oriented and comprised mostly of big functions handling a lot of special cases for different credentials and combinations.

With each new credential type, we introduced a lot of complexity to the code. New functions, conditions, and error-handling methods had to be created to make sure we would properly support the new technology without compromising our traditional authentication methods.

The following segment of code in figure 5 shows an example of the complexity introduced by adding support for a new credential type. Each new credential type requires particular functions to parse their string standards, check their validity, handle their files, etc. Below you can see we created the “token_util" library to handle these operations for JWT in GlideinWMS. Additionally, we had to add some very unique conditions, such as the one shown below. As you can see, the third “if" statement checks if the token is expired. If it is, we continue anyway. The reason for this seemingly odd behavior is to support credentials fall-backs.

The initial motivation to support credential fall-backs comes from an operations issue. In the traditional configuration model of the GlideinWMS Factory, operators specify a single authentication model required for each entry. As each entry represents a site, during the transition from X.509 proxies to SciTokens, Factory operators would have to keep track of when those sites would start supporting the new credential type to update the entry authentication method accordingly. As the SciToken adoption rate varied considerably from site to site, this quickly became a major concern. To address this issue, we implemented a temporary special

```
if "scitoken" in auth_method:
        if os.path.exists(scitoken_file):
            if token_util.token_file_expired(scitoken_file):
                entry.log.warning(f"Found frontend_scitoken
'{scitoken_file}', but is expired. Continuing")
            if "ScitokenId" in decrypted_params:
                scitoken_id = decrypted_params.get("ScitokenId")
                submit_credentials.id = scitoken_id
            else:
                entry.log.warning(
                    "SciToken present but ScitokenId not found, "
                    f"continuing but monitoring will be incorrect for
client {client_int_name}."
                    )
        else:
            entry.log.warning(f"auth method is scitoken, but file
'{scitoken_file}' not found. skipping request")
            return return_dict
```

Figure 5. Code

behavior for the "grid proxy" authentication method, which was the method used by the vast majority of Factory entries. As per the new policy, GlideinWMS Frontends would always send available SciTokens along with X.509 proxies when requesting Glideins from Factories. For this reason, when we find an expired SciToken we cannot stop the whole request, as it might still have a valid grid proxy which would succeed. Implementing this policy required adding a lot of special condition checks throughout the code.

To address the issues we have discussed and other similar problems, we decided to refactor the way GlideinWMS handles credentials. The new model consists of a new library that implements credential classes and related functions to handle secure authentication in a much more robust and streamlined way. By implementing a self-contained library, we are also able to reuse the new GlideinWMS credentials system in other Frontend-like clients for the Factory, like HEPCloud's Decision Engine.

The new library will implement the following features: abstract credential classes that can be handled generically throughout the code, revamped authentication methods that support multiple credential types and have native fallback capabilities, well-defined single purposes for credentials, and an expanded functionality of the, already present, generators.

A prototype of the library is expected by the end of 2023, to be included in the GlideinWMS development release in Q1 of 2024. The investment is considerable: for one year it will occupy almost exclusively one developer with reviews and contributions from other members of the team, which accounts for about 0.6 FTEs, one-third of our total development effort.

8 Conclusions

GlideinWMS with token and hybrid support has been running in production since the end of 2021 facilitating a smooth transition of the authentication methods used by the different collaborations. As of 2023, most VO use exclusively tokens to authorize computing. Some VO still uses X.509 to access storage.

Early adoption has its complications, mainly due to the lack of support in other tools and changing requirements, but it helps push forward the adoption in the community.

Token migration is not a drop-in replacement with isolated changes: it can be a good time to refactor the code handling credentials, but you should not stop there. The shorter life span, the spatial granularity, and the need to consider always multiple credentials will cause substantial changes in the system, may require the dynamic generation of previously static credentials, and will require an overall rethinking of all components.

9 Acknowledgements

This work was supported by the Fermi National Accelerator Laboratory, managed and operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy

References

[1] I. Sfiligoi, Journal of Physics: Conference Series **119**, 062044 (2008)

[2] I. Sfiligoi, M. Mambelli, P. Mhashilkar, D. Box, M. Mascheroni, K. Larson, B. Holzman, J. Weigand, A. Tiradani, H.W. Kim et al., *glideinwms/glideinwms: Glideinwms 3.10.5* (2023), https://doi.org/10.5281/zenodo.8383959

[3] T. Tannenbaum, D. Wright, K. Miller, M. Livny, in *Beowulf Cluster Computing with Linux*, edited by T. Sterling (MIT Press, 2001)

[4] P. Riehecky, K. Knoepfel, D. Litvintsev, P. Mhashilkar, M. Mambelli, V.D. Benedetto, P. Gartung, S. Bhat, L. Goodenough, B. Coimbra et al., *HEPCloud/decisionengine: HEPCloud decisionengine 2.0.2* (2022), https://doi.org/10.5281/zenodo.7108649

[5] *Virtual Organisation Membership Service (VOMS)*, https://italiangrid.github.io/voms, https://doi.org/10.5281/zenodo.1875371

[6] M.B. Jones, J. Bradley, N. Sakimura, *JSON Web Token (JWT)*, RFC 7519 (2015), https://www.rfc-editor.org/info/rfc7519

[7] T. Dack, M. Litmaath, F. Giacomini, H. Short, B. Bockelman, *WLCG transition from X.509 to tokens: Status and Plans* (2023), https://indico.jlab.org/event/459/contributions/11492/

[8] *INDIGO Identity and Access Management Service (IAM)*, https://indigo-iam.github.io/, https://doi.org/10.5281/zenodo.8366226

[9] A. Withers, B. Bockelman, D. Weitzel, D. Brown, J. Gaynor, J. Basney, T. Tannenbaum, Z. Miller, *SciTokens: Capability-Based Secure Access to Remote Scientific Data*, in *Proceedings of the Practice and Experience on Advanced Research Computing* (Association for Computing Machinery, New York, NY, USA, 2018), PEARC '18, ISBN 9781450364461, https://doi.org/10.1145/3219104.3219135