Message Authentication Codes for Secure Remote Non-Native Client Connections to ROS Enabled Robots*

Russell Toris† and Craig Shue† and Sonia Chernova†

***Abstract*— Recent work in the robotics community has lead to the emergence of cloud-based solutions and remote clients. Such work allows robots to effectively distribute complex computations across multiple machines, and allows remote clients, both human and automata, to control robots across the globe. With the increasing use and importance of such ideas, it is crucial not to overlook the critical issue of security in these systems. In this work, we discuss the use of web tokens for achieving secure authentication for remote, non-native clients in the widely-used Robot Operating System (ROS) middleware. Written in a system-independent manner, we demonstrate its use with an application for securing clients within the popular *rosbridge* protocol.**

I. INTRODUCTION

One of the most popular middleware suites built for robotics research and development is the Robot Operating System (ROS) [1]. ROS's robot-independent mentality allows researchers from across the interdisciplinary field of robotics to quickly make use of existing methods and plug in their own code for others to use.

At its core, ROS is a publish-subscribe (pub-sub) and message passing system which utilizes XML-RPC. This allows native clients from multiple platforms and languages to send and receive data in a peer-to-peer manner. Native clients have been written in C/C++, LISP, Python, Java, Lua, and C# (via Mono)[1].

While many times native client libraries are the best and most robust solution, they are often more heavy-weight. Emerging research in cloud robotics, web robotics, and the use of embedded devices are examples where using a more lightweight, protocol-based solution might be preferred. This allows for both smaller, optimized programs that are not required to implement the full ROS system, as is the case for an embedded devices, or efficient and diverse remote client libraries for cloud and web-based solutions (e.g., in JavaScript).

This emergence of non-native ROS clients requires some server-side node to be running within the ROS system itself. This node serves as an entry point into the system and allows clients to connect with a variety of network protocols such as HTTP, WebSockets, or plain TCP, to name a few. One such node is *rosbridge* [2]. *rosbridge* itself is simply a protocol specification for communication between a native ROS system and a remote client (either native, or non-native).

The work done with the development of ROS and remote clients has allowed for a vast array of work to be done within robotics; however, this growing popularity highlights a crucial problem with many systems to-date: the lack of proper security procedures. Many existing remote systems have utilized *security through obscurity* techniques while running live system on the Internet. However, obscurity is rarely an effective defense for valuable targets and the computer security community regards obscurity techniques as inferior to mechanisms with proven security properties, such as cryptographic primitives.

Virtual private networks (VPNs) are often used to create secure connections between the ROS system and remote clients (both native and non-native). While effective in many cases, the need to support anonymous and non-expert remote users in using of ROS systems makes VPNs less appealing due to their complex configuration and need for separate user-side software. Therefore, a bridge must be made between the robotics and security communities to address this growing problem.

In this work, we propose and develop a system-independent authentication method for remote ROS clients. The method is aimed at being used in a wide array of non-native clients and does not rely on any particular user management system. This critical point allows it to be used in the largest amount of systems and does not force researchers to adhere to, or use, particular authentication systems. Instead, we develop a schema called *rosauth* which utilizes web authentication tokensto verify remote clients via an arbitrary external user management system. We then integrate the schema as part of the *rosbridge* protocol and detail an example client use case using a web-browser based client.

II. RELATED WORK

The motivation for this work spans both the robotics and security communities. To begin, we look at additional uses of Message Authentication Codes (MACs) to solve similar security problems. Next, we look at existing work done relating directly to ROS and security. We then take a look at example work done with remote, non-native clients for use with ROS systems to emphasize the importance of security.

*This work was supported by NSF award number 1149876, *CAREER: Towards Robots that Learn from Everyday People*, principal investigator Sonia Chernova and ONR Grant N00014-08-1-0910 *PECASE: Tracking Human Movement Using Models of Physical Dynamics and Neurobiomechanics with Probabilistic Inference*, principal investigator Chad Jenkins.

†Russell Toris, Craig Shue, and Sonia Chernova are with the Department of Computer Science, Worcester Polytechnic Institute, Worcester, MA, USA {rctoris, cshue, soniac}@cs.wpi.edu

[1]http://www.ros.org/wiki/Client%20Libraries

A. MACs

The principle behind MACs is quite simple to understand. In order to ensure a message has come from a trusted source, it is hashed with a known, shared secret key using some known hashing algorithm. For example, assume client C is attempting to send a verified message m to server S. C and S both know some secret key k and agree to use a hashing function called $hash()$ and the concatenation operation "+". C will send a message with the following fields of information:

{
  mac: $hash(k + m)$
  message: m
}

When S receives the message, it will first compute $hash(k+m)$ and compare it to the received MAC. If they match, then the message received is valid. Otherwise, the message was sent by an untrusted source, was maliciously altered in transit, or was altered due to bit errors in communication. However, these guarantees only hold if a sound hashing algorithm and an appropriate length key is used [3], [4].

MACs have become essential parts of many well known and trusted security measures, including the popular IPSec (Internet Protocol Security) and SSL (Secure Sockets Layer) protocols.

While MACs typically provide message-level security, they can also be used as evidence of user authenticity. In the Kerberos protocol [5], MACs are used to create "tickets" that provide evidence that a client has authenticated using a given user or service's credentials. This approach allows an external authenticator to validate credentials and provide tickets that other remote nodes can use to ensure a user is authentic. Kerberos plays an important role in traditional computer network security, serving as part of the foundation for domain controller software, such as Microsoft's Active Directory system.

B. ROS and Security

Recent work has shed light on the many known vulnerabilities of an out-of-the-box ROS system [6]. In this work, a small robotic toy car running ROS was set up at DEF CON 20, an annual "hacking convention"[2]. The car was set up as a "honeypot" to find out what weaknesses the system had.

In particular, this work points out the use of unsecured TCP ports for ROS-to-ROS (e.g., node-to-node) communication in plain text. This allows for multiple problems such as intercepting and interpreting the plain-text messages, and the ease of spoofing messages into the system.

Additional vulnerabilities include the standard use of unencrypted data storage. While this is a valid point, this second area is not the focus of this work.

C. Non-Native Remote Clients

The emergence of non-native remote clients has opened up a wide array of research topics. One prominent use case of such clients includes utilizing this technology to enable web robotics. While porting robots to web-based interfaces has been seen in robotics for several years [7], [8], what is unique in recent trends is bridging web technologies and the power of ROS-enabled robots to bring robots to a more diverse group of non-expert users and researchers.

1) Web Robotics: As mentioned earlier, an important part of enabling this technology is a server-side node to serve as the entry point into ROS. A popular choice for this is the *rosbridge* protocol and implementation [2]. The protocol itself defines a JSON (JavaScript Object Notation) specification to gain access to the pub-sub services ROS provides. The key to *rosbridge* is the fact that clients now only have to implement a lightweight protocol as apposed to becoming a full ROS client.

One such client implementation is the ROS JavaScript library suite [9]. This library, developed as part of the Robot Web Tools effort [10], allows web browsers to communicate with and visualize data to and from ROS. These libraries communicate to *rosbridge* via WebSockets, which is built on top of HTTP. This allows remote users, both expert and non-expert, to gain access to robots remotely using a robust, cross-platform solution.

In [11], an entire remote lab was developed for a PR2 robot. This robot, running ROS and *rosbridge*, was accessible across the web and could be fully controlled using a modern web-browser. Not only was this a huge step forward for enabling researchers to gain access to robots they themselves may not have access to in their own labs, but it also allowed for work to be done quickly across the web using remote users [12], [13].

2) Cloud Robotics: While web-browser-based approaches have become a popular use of non-native ROS clients, examples exist in other areas as well. Cloud robotics is another emerging field that has made use of such non-native ROS clients. One such project, known as Rapyuta[3], has developed a platform as a service (PaaS) framework for robots using remote, elastic, cloud-based ROS compatible computing environments. A JSON based specification similar to *rosbridge* was developed to connect the remote, cloud processes to the core ROS system.

While the use of non-native clients has opened up the research field to a vast array of new possibilities, the notion of security has been dramatically overlooked. By creating entry points into the ROS system, such as *rosbridge*, we exacerbate the problems outlined in [6]. In order to allow remote users to connect, we now create open, unsecured TCP ports with a clearly defined, lightweight protocol to control an entire robotic system. With the gaining momentum of these technologies, it is critical to begin to solve the issues associated with doing so.

III. GOALS

We have several goals in creating our secure environment:

[2] http://www.defcon.org

[3] http://www.rapyuta.org

1) The main ROS core, associated native ROS clients (including the robot itself), and non-native clients must be able to communicate with guaranteed message authentication, integrity, and confidentiality.
2) Any remote host must be able to request service from the system.
3) An arbitrary external authentication system is used to provide credentials for internal ROS devices.
4) The system must provide proper authenticity and connection identity, even when multiple clients are multiplexed to the same source IP address (such as when a network address translation (NAT) device is used).
5) Connections that do not successfully authenticate will be terminated.

By meeting these goals, a system can ensure ROS devices can communicate privately, without concern for tampering or alterations from malicious outsiders.

IV. METHODOLOGY

Our security goals implicitly separate the requirements for individual packets from session security. An existing security protocol, such as IPSec or SSL, can be used to create secure tunnels for communication. VPN software is often used for this purpose to allow ROS-enabled components to communicate with each other. However, VPNs often require each party to install specialized software and carefully configure settings. While useful in closed networks, these limitations may hinder participation of external users on non-native clients.

Rather than using VPNs, we will use the SSL protocol to ensure confidentiality, integrity, and authenticity of individual packets. By using certificates issued by trusted certificate authorities, SSL can ensure that external clients know each ROS system, including the external authenticator, is legitimate. SSL also uses MAC and encryption algorithms to provide message integrity, authenticity, and confidentiality. In Figure 1, we depict a network that supports both VPN and SSL connectivity for remote users. We use a firewall to block all traffic that does not use either the VPN or authorized SSL connections.

A. Authenticating Remote Users

While SSL provides a secure channel for communication, it does not provide any assurances that a connected client is an authorized user. Instead, a system, which we call the External Authenticator (the top-right machine in Figure 1), must be responsible for user authentication. This server, based off the Kerberos protocol, must maintain a list of users, their associated access levels, and the private credentials, such as a password, associated with each user account. To establish a connection, a client must approach the External Authenticator with the appropriate credentials that the External Authenticator can use to verify the legitimacy of the client's identity claim (e.g., a username and password). If the claim is valid, the External Authenticator will provide client with a token, which serves as identifying evidence to other systems, that the client can use to gain access to other systems. This approach allows a single server to take on the burden of user authentication while allowing other systems to seamlessly use this identity.

The External Authenticator must carefully construct the security token to prevent attackers from creating counterfeit tokens or altering existing tokens. The token in the authenticator must include:

- `client` (string): The client string contains the IP of the client where this message originated.
- `dest` (string): The destination string contains the IP or host of the server the client is trying to connect to.
- `rand` (string): A random string is added to the hash as a nonce to prevent replay attacks and cookie stealing, and allow for multiplexing (explained further in this section).
- `t` (int): A count of seconds since the start of the Unix epoch is given, indicating the time the original MAC was created.
- `level` (string): A user level string is provided to state what level of user is connected (e.g., admin).
- `end` (int): An end time in seconds is given stating how long the client is authorized to remain connected.

The `rand` value plays several important roles. First, the random value serves as a nonce, indicating that the same random value should not be used multiple times in a given time period. This ensures an attacker cannot simply replay a prior request to be authenticated or steal another client's credentials. This value also enables servers to demultiplex multiple clients that happen to share an IP address (such as those behind a NAT device). Finally, this value must be kept secret to ensure other clients cannot present the token. However, since the entire communication is protected by an SSL tunnel, an adversary would be unable to obtain this value through eavesdropping attacks.

The External Authenticator will use a delimiter and concatenate these fields, in the indicated order into a string, `token_fields`. The authenticator will then produce a hash MAC using a secure MAC function, such as one of the SHA-2 algorithms [14]. The hash MAC will be constructed using `MAC = hash(key + token_fields)`. The External Authenticator will then use a delimiter and concatenate both the `token_fields` and `MAC` fields into a single string, `token`. This token is then provided to the client, allowing it to attest to its identity with supporting evidence.

When the client contacts a server with a token, the server will perform several checks on the token and the client, as outlined in Figure 2. This check is made during the connection establishment phase and the connection is aborted if any of the checks fail.

By using SSL, we need only perform user authentication once per connection. SSL's protection of the connection makes it impractical for an attacker to inject commands into a protected connection or to assume control of another party's session. This allows the server to authenticate a connection with only a single packet from the client.

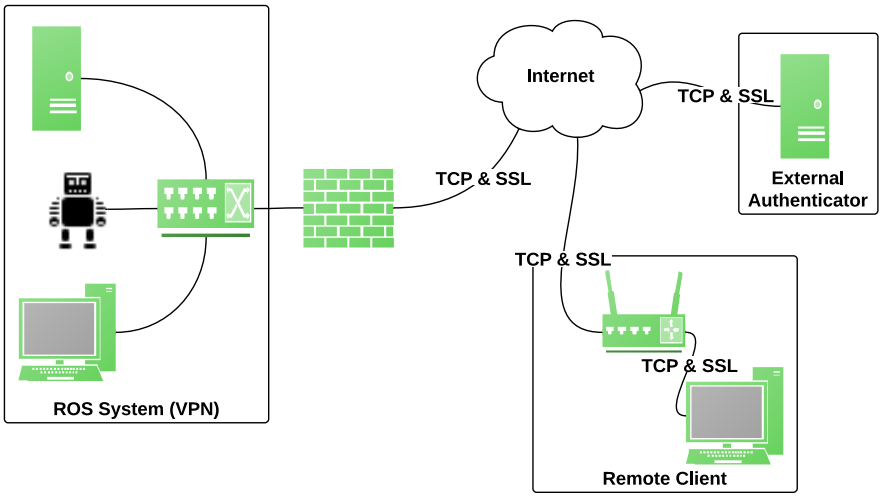

Fig. 1. **A high-level outline of the developed method. To the left, the ROS system (including its associated robot) is protected by a firewall and only accessible through a VPN or a single port listening for SSL traffic. Remote clients (bottom) can request the authentication token and associated fields from the external authenticator (top-right) in order to attempt an authenticated connection.**

1: **procedure** CHECKAUTHENTICATION($mac, data$)
2: **if** $sha512(key + data)$ is not mac **then**
3: return $False$
4: **else if** $data['client']$ is not socket's client IP **then**
5: return $False$
6: **else if** $data['host']$ is not server's IP **then**
7: return $False$
8: **else if** $data['t']$ is not current time $\pm\delta$ **then**
9: return $False$
10: **else if** $data['end'] \leq$ current time **then**
11: return $False$
12: **else**
13: return $True$
14: **end if**
15: **end procedure**

Fig. 2. **The authentication check procedure used to verify connection requests from non-native ROS clients. δ represents a small amount of time to account for loosely synchronized system clocks.**

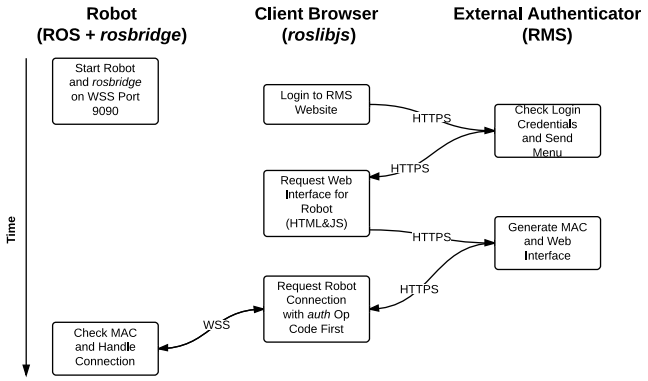

Fig. 3. **A high-level pipeline showing the flow in which a remote client can securely connect to the robot.**

V. APPLICATION

We now present an example use case and application of the above methods. In this application, we consider the case of web-browser-based clients connecting to a ROS-enabled robot. The techniques used in this application are robot independent and have been tested to remotely control multiple robots, including the Willow Garage PR2 and KUKA youBot. For this application, our clients are using a web-browser based interface designed with *roslibjs*[4] and connect to a *rosbridge* server using Secure WebSockets (WSS), which is built on top of HTTPS. Furthermore, the external authenticator is the Robot Management System (RMS)[5] which contains its own user management [13]. Internal authentication is done via the *rosauth* package[6] we created.

A. Authentication Pipeline

We begin by looking at a high-level pipeline of the application flow. This serves to give a general idea of how the system functions before we take a closer look at some of the pieces. This pipeline is depicted in Figure 3.

We start at the top left of the diagram (note that time progresses downwards). To begin, the robot is brought up and a *rosbridge* server is started on the robot. This server runs on port 9090 and uses a Secure WebSocket with a signed SSL certificate. At this point, the robot is fully up and running, and *rosbridge* is listening for any connection.

Next, a user from an external Internet location opens up a web browser and connects to a running instance of the RMS. In our case, the RobotsFor.Me[7] site was used [13]. The connection to this site is made via HTTPS (again, using

[4]http://www.ros.org/wiki/roslibjs
[5]http://www.ros.org/wiki/rms
[6]http://www.github.com/WPI-RAIL/rosauth
[7]http://www.robotsfor.me

a signed SSL certificate). The user logs into this site using their login credentials which are verified by the RMS (i.e., external authenticator) itself. Once logged in, the client is redirected via a HTTP *REDIRECT* (status code 302) to the main menu page. This page provides the user with a list of available robots and interfaces.

Upon receiving the menu, the user can click on the available robot and a request is sent to the RMS for an interface to control this robot. An example interface is shown in Figure 4 to control the Willow Garage PR2 robot.

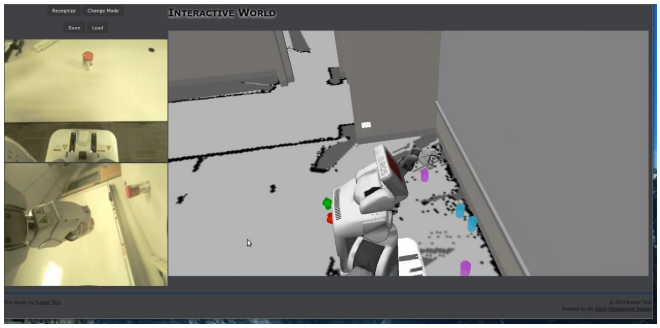

Fig. 4. **An example web-interface that uses a non-native ROS client library to control a ROS enabled robot.**

Now, the RMS is responsible for supplying the client with the appropriate connection information and MAC string. A deeper discussion of this is explored in Section V-B. Once this information is generated, it is sent back to the user as an HTML and JavaScript page via HTTPS.

At this point, the client now has an HTML page that is filled in with the appropriate connection information. The browser will execute the generated JavaScript and open a Secure WebSocket connection to the ROS system. Here, it passes along the supplied authentication information and the ROS system decides if the information is valid (discussed in Section V-C). Once verified, the connection is kept open and data can stream between the client and the robot.

B. Client-Side Credential Acquisition

During this process, the RMS is responsible for supplying a correct MAC and associated information so that the ROS system will authenticate the connection request. To do so, it first looks internally at its own secured database to acquire the secret key. This key, a string of 16 characters, is randomly generated upon installation of the RMS and has also been installed on the ROS server. Next, it gathers the rest of the information by checking information such as the client's IP address (which cannot be spoofed due to the TCP hand-shaking process) and generates a random string. In order to determine the end time, the RMS looks internally at its user database. In this instance, admins were authorized to connect to the robot at any time, and general users were restricted to a specific time frame. This information was determined server-side and sent back to the client and included in the MAC.

All the appropriate information is hashed together using SHA-512 and is filled in as JavaScript that will be sent back to the client. This JavaScript serves as the client's security token.

C. Server Side Authentication

In order to incorporate the methods developed in this work into the ROS system, the *rosbridge* protocol itself was modified. An additional op code, `auth`, was added to the protocol with the following JSON definition:

```
{
  "op": "auth",
  "mac": <string>,
  "client": <string>,
  "dest": <string>,
  "rand": <string>,
  "t": <int>,
  "level": <string>,
  "end": <int>
}
```

Now, if authentication is enabled on the *rosbridge* server, it will wait for this op code to come in before accepting any ROS messages.

When the request does come in, a ROS service call is made to the *rosauth* node. This generic node takes in the above information and verifies it according to the procedure outlined in Figure 2. The genericness of this node allows these methods to be used in additional server implementations outside of *rosbridge*.

Once the request is checked in *rosauth*, a true or false response is sent back to *rosbridge*. Any false response is treated as an invalid request and results in a severed connection.

D. Unauthorized Connection Attempts

While the above tests showed the pipeline for a successful connection, it was also critical to test against unauthorized requests. Several tests were made to verify this.

In the first test, an unauthorized instance of RMS was set up to try and connect its own clients to the robot. In this case, all information in the MAC would be valid; however, the secret keys would mismatch. As expected, the connection was immediately dropped.

In additional tests, changes were made to certain parts of the MAC information that did not match the original information used to generate the MAC. This too lead to an appropriately severed connection. In all known previous work, these unauthorized connection attempts would be unconditionally accepted allowing full control of the robot to potentially malicious users.

VI. CONCLUSION

In this work, we have explored and developed a custom MAC based authentication schema for remote, non-native ROS clients. As expressed in [6] and demonstrated at DEF CON 20, the need for security techniques must be brought into the robotics and ROS communities.

While we note that some of the methods used in this work could solve broader problems within ROS (discussed further

in Section VII), we focus on the case of having remote, non-native ROS clients. For native ROS clients, we assume such devices can communicate directly to each other via some trusted, secure network.

This work provides new methods for providing security measures aimed at authenticating remote users from any IP address using non-native ROS clients. As exemplified in work such as [9], [11], [2], [12], [13], the ability to utilize these non-native clients will allow roboticists to utilize a wider range and more diverse group of users and researchers. Nevertheless, with the growing momentum behind such easy to use lightweight protocols and techniques, it becomes ever more important to ensure these environments are run in a safe and secure manner.

The developed security token schema ensures that only clients which have been authenticated from some trusted external authentication source are allowed access to the robot. The development of such a generic schema also allows for a wide array of out-of-the-box (such as the RMS) systems, or custom user management systems to be used to control access to the robot. Furthermore, the *rosbridge* protocol has been further developed and modified to incorporate the developed methods.

VII. FUTURE WORK

Even with the methods developed in this work, there is still room for a vast array of work to be done in this growing area. To begin, we look again at the case of non-native clients.

While this work provides solutions to authentication, another problem with the system itself is *authorization*. Many times, we do not want to allow certain remote clients access to the entire ROS system. This allows users to send direct commands to the robot which may bypass certain safety constraints. The schema developed in this work provides the ability to expand upon this idea.

Within the security token is a `level` field. This field is associated with the user level of the current client. This arbitrary string is determined by the external authenticator (allowing for a variety of different levels), but the developed authentication schema ensures the client did not tamper with this user level. Thus, an authorization node could be put in place which is configured as a list of ROS topic and service names and the associated user level. If a client attempts to publish or subscribe to a data stream that they are not authorized to access, the request could be ignored.

One additional area of future work involves incorporating the techniques developed in this work for native ROS clients. In this work, we assumed core ROS systems and clients could be wrapped in a VPN; however, as stated earlier, this may not always be possible or desired. The developed technique thus could be adapted into the core ROS system to authenticate any and all *native* ROS clients. The abstractness of the *rosauth* package also makes this a more feasible solution.

ACKNOWLEDGMENT

The authors thank Willow Garage, Inc. and Prof. Chad Jenkins of Brown University for their support and development of *rosbridge* and its underlying protocol to incorporate the techniques developed in this work.

REFERENCES

[1] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. B. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[2] C. Crick, G. Jay, S. Osentoski, and O. Jenkins, "ROS and rosbridge: Roboticists out of the loop," in *th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, March 2012, pp. 493 –494.

[3] X. Wang and H. Yu, "How to break md5 and other hash functions," in *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, ser. Lecture Notes in Computer Science, vol. 3494. Springer, 2005, pp. 19–35.

[4] X. Wang, Y. L. Yin, and H. Yu, "Finding collisions in the full sha-1," in *Proceedings of the 25th annual international conference on Advances in Cryptology*, ser. CRYPTO'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 17–36.

[5] J. G. Steiner, C. Neuman, and J. I. Schiller, "Kerberos: An authentication service for open network systems," in *Usenix Conference Proceedings*, 1988, pp. 191–202.

[6] J. McClean, C. Stull, C. Farrar, and D. Mascareas, "A preliminary cyber-physical security assessment of the robot operating system," in *Proceedings of the SPIE: Unmanned Systems Technology XV*, vol. 8741, May 2013.

[7] K. Taylor and J. Trevelyan, "A Telerobot On The World Wide Web," in *1995 National Conference of the Australian Robot Association*. Melbourne: Australian Robotics Association, July 1995.

[8] K. Goldberg, Ed., *The Robot in the Garden: Telerobotics and Telepistemology in the Age of the Internet*. Cambridge, MA, USA: MIT Press, 2001.

[9] S. Osentoski, G. Jay, C. Crick, B. Pitzer, C. DuHadway, and O. Jenkins, "Robots as web services: Reproducible experimentation and application development using rosjs," in *Proccedings of the 2011 IEEE International Conference on Robotics & Automation*, 2011.

[10] B. Alexander, K. Hsiao, C. Jenkins, B. Suay, and R. Toris, "Robot web tools [ROS topics]," *Robotics Automation Magazine, IEEE*, vol. 19, no. 4, pp. 20 –23, December 2012.

[11] B. Pitzer, S. Osentoski, G. Jay, C. Crick, and O. Jenkins, "PR2 remote lab: An environment for remote development and experimentation," in *2012 IEEE International Conference on Robotics and Automation (ICRA)*, May 2012, pp. 3200 –3205.

[12] S. Osentoski, B. Pitzer, C. Crick, G. Jay, S. Dong, D. H. Grollman, H. B. Suay, and O. C. Jenkins, "Remote robotic laboratories for learning from demonstration - enabling user interaction and shared experimentation," *International Journal of Social Robotics*, vol. 4, no. 4, pp. 449–461, 2012.

[13] R. Toris and S. Chernova, "RobotsFor.Me and Robots For You," in *Interactive Machine Learning Workshop, Intelligent User Interfaces Conference*, March 2013, pp. 10–12.

[14] S. Gueron, S. Johnson, and J. Walker, "Sha-512/256," in *Proceedings of the 2011 Eighth International Conference on Information Technology: New Generations*, ser. ITNG '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 354–358. [Online]. Available: http://dx.doi.org/10.1109/ITNG.2011.69