CERTIFICATE BASED STRONG CLIENT AUTHENTICATION AS A REPLACEMENT FOR USERNAME/PASSWORD

LAWRENCE E. HUGHES

CTO, SIXSCAPE COMMUNICATIONS

FOR ROOTCON X, SEPT 2016

THE PROBLEM WITH USERNAME/PASSWORD AUTHENTICATION - USERNAMES

- *Usernames* are fairly easy to create and remember, but if they are not qualified with an Internet domain name (like E-mail usernames) and they are for a popular service (e.g. Skype) it can be difficult to get your preferred username. You can wind up with unwieldy names like *larry4363*. If the username includes an Internet domain name (e.g. larry@hughesnet.org) then you can greatly reduce the problem of collision with other users (especially if you can use any domain name). You need only find a userid that is unique within a specific domain.
- That being said, even though you can easily change your password, changing your username may be very difficult or impossible with most online services, especially if you want continuity with past usage of that service. And your username may be exposed (e.g. in mail headers).

THE PROBLEM WITH USERNAME/PASSWORD AUTHENTICATION - PASSWORDS

- The real problem is with *passwords,* and with the combination of a username and a password for authentication to an online service. This scheme is pretty much completely broken. **We can’t use it anymore.**
- Humans are *notoriously* bad at coming up with strong, but easy-to-remember passwords. And in theory you should use a different password on *every* online service, or if a hacker cracks your credentials on one service, they will try it on many others, often with success.
- To complicate it further, you should change your passwords periodically (and some systems force this, and will even keep a tail of most recently used passwords, which you can’t reuse), making it even more difficult to keep track of them.

THE PROBLEM WITH USERNAME/PASSWORD AUTHENTICATION - HACKING

- I can hack into an online service and grab their username/password database. A good site will have protected that with hashing and salting (e.g. PBKDF2) as described here:

  https://crackstation.net/hashing-security.htm

- All that hashing and salting will just slow me down. It won't stop me from harvesting all those credentials. This has happened to Target and many other online vendors. As long as username/password authentication is used, there is no real defense against mass credential harvesting.
- If you send your username and password in clear text, I can see with WireShark!
- I once created a hacking tool that took the output of a sniffer, extracted all packets for port 110 (POP3) and 143 (IMAP), put them together like TCP, and wrote them to a file. I left it running in our office all day, and wound up with a list of everyone's email, including their usernames and passwords. We switched to POP3S and IMAPS immediately.

THE PROBLEM WITH USERNAME/PASSWORD AUTHENTICATION - CRACKING

- You can download the 10,000 most commonly used passwords from various places. Some 97% of all user-generated passwords will be among the first 1,000 of those. This helps cracking hashed passwords, even with salt.
- If my clever online system forces you to choose a really complex password (at least one upper case, one lower case, one number and one special character, no part of your username, no English words, no recently used passwords, etc), then chances are good you will never be able to remember an acceptable password and will *write it down*. Probably on a PostIt note under your keyboard. This is an *unusually* bad idea.
- The strength of a password is almost entirely determined by its *length*, not its complexity. The strength of a password is proportional to m to the n-th where m is the size of the charset (e.g. 26 for only lower case, 52 for upper or lower case) and n is the length of the password.

THE PROBLEM WITH USERNAME/PASSWORD AUTHENTICATION – KEYBOARD LOGGING

- I can easily put a Trojan Horse on your computer and watch everything you are typing. This is difficult (but not impossible) to scale up to millions of users.
- You can create *incredibly* strong passwords by taking two names or English words and separating them with a digit or special character. If you slightly misspell one or both words, that makes it even more difficult to crack. Try to exceed 14 characters total. For example, *Raspberri*Sherbet* – is *very* strong, and yet that is really easy to memorize.
- How strong? The password “y6&p2G” has a strength of 72 to the 6th (about 1.39 E+11). The password “Raspberri*Sherbet” has a strength of 62 to the 17th (about 2.95 E+30), which is some 21 quintillion times stronger. The complex but short one can be cracked in seconds.
- But if I’m watching your keystrokes with a logger, even if you use a strong password like that, and you are connecting to a site with TLS v1.2 using RSA2048 bit and AES256, you are still *dead meat*. I’ve got your credentials. *All your bases are belong to me!*

SO WHAT IS THE SOLUTION?

- The solution involves *cryptography*. And not just any old cryptography (like AES), but *asymmetric key* (public/private key) cryptography. Like RSA or ECC. The fun stuff. The stuff that makes SSL/TLS work. The same stuff that makes S/MIME work. And digital certificates. And PKI. It's actually part of SSL/TLS since SSL 3.0 (surprise!)
- Symmetric key cryptography uses the same key to encrypt and decrypt. Intuitive.
- Asymmetric key cryptography uses a matched pair of keys, a *public key* and a *private key*. You encrypt with either of them, but you can only decrypt with the *other* key of the pair (*really* non-intuitive).
- PKI embeds the *public* key in a digital certificate that has info needed to verify it's really the public key of the person it claims to be for. And publishes it in a directory for anyone to use. The *private* key is best kept in a crypto token and *never even loaded into your computer*.

SO HOW DOES SSL/TLS WORK?

- There are three phases in an SSL/TLS handshake.
  - 1 – the server authenticates itself to the client (I'm really the server for *amazonwomen.com*)
  - 2 – the client authenticates itself to the server (I'm really *Lawrence Hughes from Sixscape*)
  - 3 – the client creates a symmetric session key and securely shares it with the server, which will be used to encrypt everything after the handshake, in both directions
- Today, the client to server authentication usually happens *after* the TLS handshake, in the encrypted tunnel, using username/password authentication. We want to use the *real* phase 2 which happens *during* the TLS handshake, before the application protocol begins. Not many people use that today. It's been around since SSL 3.0. But it requires *Client Digital Certificates*.

WHAT IS A SERVER DIGITAL CERTIFICATE?

- You are probably familiar with *SSL server certificates*. You can buy them from online CA's like DigiCert, or get them for free from the Encrypt Everything project. Or create your own with OpenSSL (those are not *trusted*, or "public hierarchy" certs – they are like counterfeit money).
- It is a digital document that binds the server's public key to identifying information, like the server's FQDN (*www.amazonwomen.com*); who issued the server cert (digicert.com); an expiration date; and various other information. The standard involved is X.509 v3 (RFC 5280). That binding is done by the CA (a trusted third party) digitally signing the certificate.
- Using the information in the cert I can check if it is trusted (it chains up to a trusted root cert); whether it is currently valid (or has expired); and its revocation status (whether the issuing authority has revoked it or not). Anyone presented with a server cert should check all of those things.
- A secure server only needs one server cert for any number of users.

Demo of Server Certificate

HOW DOES SSL/TLS PHASE 1 WORK?

- At the start of the TLS handshake, the client and server negotiate various things.
- Then the server sends its Server Certificate to the client.
- The client checks the server cert trust, validity and revocation status (abort if failed)
- The client creates a short random string, encrypts it with the server's public key (from the server cert) and sends the result to the server as a *crypto challenge*
- The server decrypts the crypto challenge with its private key and returns the result to the client as the *challenge response*
- If the challenge response matches the original string, that proves the server has the private key corresponding to the public key in the server cert, without actually revealing that key
- That gives you strong *server to client authentication* (Phase 1 success)

Demo of Crypto challenge

WHAT IS A CLIENT DIGITAL CERTIFICATE?

- A Client Certificate is like a Server Certificate, but it has different information associated with the public key.
- It is a digital document that binds the client's public key to identifying information, like the user's name and email address; who issued the server cert (Comodo); an expiration date; and various other information. The standard involved is still X.509 v3 (RFC 5280). That binding is done by the CA (a trusted third party) digitally signing the certificate.
- Using the information in the cert I can check if it is trusted (it chains up to a trusted root cert); whether it is currently valid (or has expired); and its revocation status (whether the issuing authority has revoked it or not). Anyone presented with a client cert should check all of those things.
- *Every user* of a secure server requires a distinct client cert that identifies *them*. WHOA.

Demo of Client Certificate

HOW DOES SSL/TLS PHASE 2 WORK?

- If SCA is configured, after Phase 1 is complete, the server asks the client for a client cert.
- The client lets the user select a client cert from the available certs and sends it to the server
- The server checks the client cert trust, validity and revocation status (abort if failed)
- The server creates a short random string, encrypts it with the client's public key (from the client cert) and sends the result to the client as a *crypto challenge*
- The client decrypts the crypto challenge with its private key and returns the result to the server as the *challenge response*
- If the challenge response matches the original string, that proves the client has the private key corresponding to the public key in the client cert, without actually revealing that key
- That gives you strong *client to server authentication* (Phase 2 success)

HOW DOES SSL/TLS PHASE 3 WORK?

- After all the authentication is done, the client creates a symmetric session key (and IV) for the negotiated symmetric key algorithm
- The client then encrypts the symmetric key and IV with the server's public key (from its server cert) and sends it to the server
- The server decrypts the encrypted session key and IV with its private key
- Voila - A symmetric session key has been securely shared – Phase 3 Success
- After that everything sent by either end is encrypted by that key, and the other end receives the encrypted traffic and decrypts it with the same symmetric session key
- A new session key is generated for every SSL/TLS session – it can even be periodically rekeyed

Demo of strong client auth

SO HOW DO I GET A CLIENT CERT?

- Some CA's also provide client certs, like Comodo, DigiCert, etc
- There are free ones that only authenticate your e-mail address
- There are ones for sale that validate your name, e-mail address, company, etc
- Sixscape has created a CA (IDCentral) that lets you request and download client certs via a secure protocol (*Identity Registration Protocol*, or IRP, port 4604). It also allows you to download the CA Certs, determine revocation status, etc, all via IRP.
- IRP-enabled clients can let you do this via a GUI, or it can be partially or fully automated, to hide the complexity of PKI from the user. The client can also publish the client cert in LDAP, or obtain other user's client certs from LDAP.

WHAT IS THE BLACKBIRD SECURE E-MAIL CLIENT?

- Since Snowden's revelations we know that all US software has weakened crypto algorithms and/or backdoors.
- Sixscape has created a new secure E-mail client called *Blackbird* (as in SR-71) with trusted crypto, IRP integration (with our IDCentral CA), Personal Address Book, LDAP/AD integration (as a Group Address Book), MS Certificate Database manager, Crypto Token Manager, etc.
- It can create S/MIME secured messages (with attachments) and send them via SMTP, via FTP/S or via SFTP. It can also save them to an .eml file (e.g. in DropBox, OneBox). The recipient can double click on the file which pops the S/MIME message into their secure E-mail client.
- It is dramatically easier to configure and use S/MIME with Blackbird than with Outlook.
- We have extended S/MIME to other protocols just like SSL/TLS was earlier.
- Next we're going to create an XMPP+S/MIME end2end secure chat app.

Demo of blackbird Secure E-mail Client

Lockheed SR-71 Blackbird

HOW DO I PROTECT MY PRIVATE KEY?

- By default, when I create a keypair (when requesting a client cert), the private key is created on my computer, and stored in my MS Certificate Database. It can be protected with no, weak or strong protection. Strong protection requires me to enter a passphrase each time I use that private key. It could still be hacked.
- It's better to keep your private key in a FIPS-140-2 crypto token. You can import key material into a token, or you can actually create the key material inside the token. Once the private key is in the token, there is no way to get it out, but you can *use* it via PKCS 11.
- A crypto token can look like a thumbdrive, or a smartcard (with contacts or contactless).
- When you plug in a crypto token, you must supply a passphrase to allow access to the private key(s) in it. Once supplied, the key material *appears* to be in the MS cert DB, but isn't really.

WHAT IS A CRYPTOGRAPHIC TOKEN?

- A cryptographic token is a small device with limited memory (maybe 64K bytes), a small CPU, firmware that can do asymmetric key (RSA), symmetric key (AES), message digest (SHA) and PKCS11 protocol, and typically a USB interface. They cost about US$15 in single quantity.
- The private key can be loaded into a token (e.g. from PKCS 12 file) or created inside it. Once inside the token, the private key can never leave (it's a *Hotel California* for private keys).
- You transmit things to encrypt or decrypt into the token (using PKCS 11 via USB), do the encryption or decryption in the token, then retrieve the results (using PKCS 11 via USB).
- If a private key is ever kept on your computer's file system, or is used in your computer's memory (with virtual memory swapping going on), there are attacks to get the private key.
- With a cryptographic token, the private key is never really *in* your computer.

Demo of crypto token

WHAT PLATFORMS CAN I USE SCA ON?

- Anything that uses SSL/TLS can in theory do SCA (need client certs)
- Username comes from client cert, no password or 2FA needed (crypto challenge far stronger)
- You can use it with Web Apps, Window Apps or Mobile device apps.
- It can be used for authentication to SMTPS, IMAPS, LDAPS, etc (server must support)
- SCA is the future of authentication for all applications.
- With private keys kept in hardware crypto token, this is military grade security.
- No need for username/password database on server
- Hacker could see everything I'm typing, sniff all network traffic, cannot assume my identity!
- Without my private key (and passphrase), they cannot assume my identity.

IN CASE YOU ARE WONDERING WHAT INSPIRED OUR NAME AND LOGO…

Netscape Communications made enormous contributions to the IPv4 Internet with the first viable web browser, first viable web server, first LDAP server, SSL and many other innovations.

Sixscape Communications is making the same kind of contributions to the IPv6 Internet, with IRP (first Certificate Management Protocol with authentication), authenticated IP address registration, extending S/MIME to protocols other than SMTP, and *End2End Direct* connectivity.

Thank you For listening

"The right to buy information weapons is the right to be free“
A.E. van Vogt, *The Weaponshops of Isher*, updated

Sixscape – Purveyors of Fine Information Weapons since 2014

www.sixscape.com

Privacy is a Fundamental Human Right