Bolt on some Crypto

Michael Samuel @mik235 https://miknet.net/

Ruxcon 2014

Securing The Network - TLS & SSH

IETF Standards:

SSH - RFC 4250-4255

- Remote shell
- File transfer
- TCP port forwarding, socks proxy
- Pipe commands over ssh (stdin/stdout)
- Originally a replacement for BSD r-commands

TLS - RFC 5246.

- https://
- Optional for SMTP, IMAP, POP3, XMPP, LDAP

TLS & SSH - Cryptographic Services

- Authentication
  - more to come on this...
- Integrity
  - Any tampering with the connection will be detected
  - Limitation: attacker can drop the session
  - Limitation: DoS
- Privacy
  - Cannot see contents of session
  - Limitation: traffic analysis (aka metadata)

Public Key Cryptography Primer

Keypair:

Private Key - This key must be kept safe! *Don't email me your private key!*

Public Key - This key can be shared with anyone you need to communicate with

Signing:

The Private Key is used to sign a hash of a message, which can be verified by anyone with the public key

Encryption:

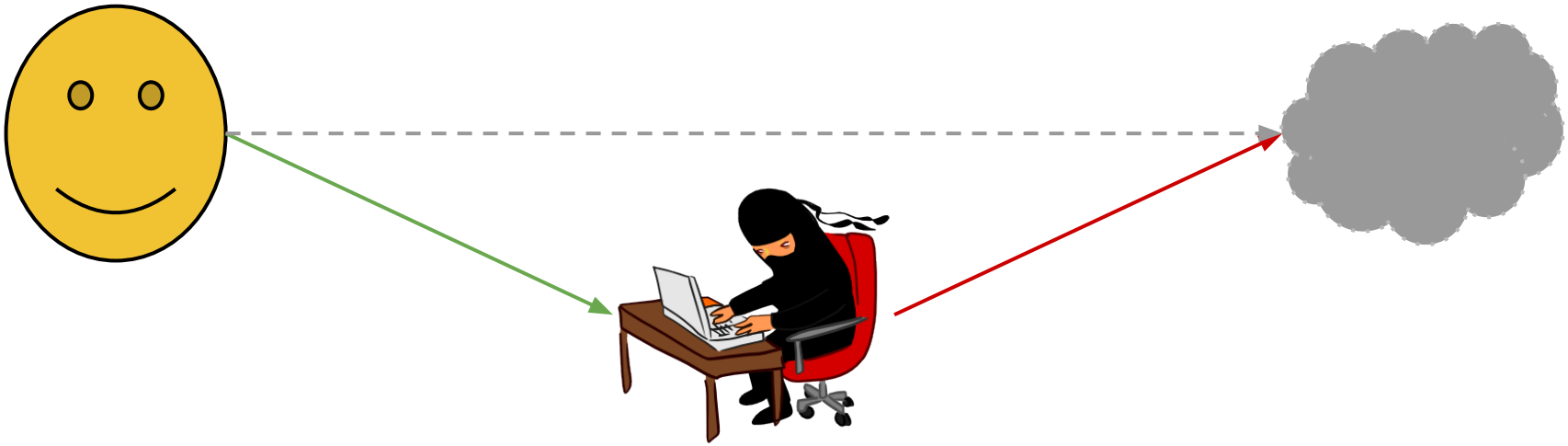
The Public Key is used to encrypt a message, which only the holder of the Private Key can decrypt

MiTM Attack

"Man-in-the-middle attack"

(The actual attack isn't gender specific)

1. Intercept client connection and answer like a server.
2. Connect to the real server (optional)
3. Log or modify data as it passes through



https://openclipart.org/detail/151741/ninja-working-at-desk-by-hector-gomez

MiTM Attack - Linux Quickstart

- `iptables -t nat -A PREROUTING -p tcp --dport 5222 -j \`
  `REDIRECT --to-port 5002`
- Run your client program, listening on 5002
- Route the traffic *through* your linux box using arpspoof, routing protocol
- If using dns spoofing, IP tables not required

To get the original dest IP:

In C:

```
getsockopt(s, SOL_IP, SO_ORIGINALDEST, &addr, &addrlen);
```

In Python:

```
packedDest = s.getsockopt(socket.SOL_IP, 80, 16)
(destPort, ) = struct.unpack(">H", packedDest[2:4])
destHost = socket.inet_ntoa(packedDest[4:8])
```

SSH Host Keys

OpenSSH caches host keys:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

StrictHostKeyChecking - on seeing a new host:

- yes - don't connect
- no - cache host key and connect
- ask (default) - display the fingerprint and ask user

ssh-keyscan can collect host keys from remote systems - allows you to pre-populate known_hosts

SSH Host Keys - API gotchas

JSch:

`StrictHostKeyChecking=no` won't cache the host key!

Paramiko:

```
client.load_system_host_keys()
client.set_missing_host_key_policy(paramiko.RejectPolicy)
```

`paramiko.WarningPolicy` won't cache the host key!

Just pre-populate /etc/ssh/ssh_known_hosts if using APIs - no need for write access to known_hosts

SSH MiTM - you can do ‘em

Often `StrictHostKeyChecking=no` is set on servers with unattended ssh sessions

- rsync jobs
- remote commands

OpenSSH still connects if the host key changed and you’re using public key authentication!

A MiTM server could just accept pubkey auth for any key (without knowing the key).

SSH Client Authentication

You can create a client keypair with `ssh-keygen`, then add it to `~/.ssh/authorized_keys` on remote hosts. This can be put in kickstart/preseed files.

Even if the remote server is compromised your private key should be safe, so you don't need a fresh one for each server you connect to.

You can do "two-factor" in OpenSSH with the `AuthenticationMethods` sshd_config option.

TLS - X.509 Certificates

Certificate chain from my website:

```
 0 s:/CN=www.miknet.net
   i:/CN=StartCom Class 1 Primary Intermediate Server CA
 1 s:/CN=StartCom Class 1 Primary Intermediate Server CA
   i:/CN=StartCom Certification Authority
```

`StartCom Certification Authority` is trusted by my system

Subject: the entity identified by the certificate

Issuer: the authority that signed the certificate

Domain Validated: demonstrated control of the domain to CA

Extended Validation: demonstrated that you are the organisation and domain holder in the certificate

The unverified certificate

A Root CA is just a self-signed certificate

Intermediate CAs and the certificate are signed by their parent CA

You can create an entire *unverified* chain using the openssl command line. Only the public key matters.

Even the most diligent support staff would tell users to click through.

Dialogs that shouldn't exist

WARNING: TLS APIs suck

There are 3 types of TLS APIs:

- Go verify the certificate yourself
  - Generally OpenSSL or wrappers
- What's a certificate?
  - High level abstractions over OpenSSL written by programmers who don't know/understand
  - all of the Python 2.x standard library
- We do what a web browser would
  - These are rare - python-requests.org, libcurl

TLS - Verifying the hostname

Most TLS libraries do not check that the certificate matches the hostname - even if you turn on verification.

Should you trust a certificate for www.miknet.net when accessing your online banking?

The hostname must match either the **CN** field or one of the **SubjectAltName** extensions.

WARNING: NULL bytes are valid

Match the name the user requested, not DNS SRV/MX

STARTTLS

```
<?xml version='1.0' ?>
<stream:stream to='jabber.org' xmlns='jabber:client' xmlns:stream='http://etherx.jabber.org/streams' version='1.0'>
<?xml version='1.0'?>
<stream:stream xmlns='jabber:client' xmlns:stream='http://etherx.jabber.org/streams' from='jabber.org' id='5ce74cfce8e91fc4' version='1.0'>
  <stream:features>
    <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
    <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
      <mechanism>DIGEST-MD5</mechanism>
      <mechanism>PLAIN</mechanism>
    </mechanisms>
  </stream:features>
  <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
  <proceed xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
```

STARTTLS removed

```
<?xml version='1.0' ?>
<stream:stream to='jabber.org' xmlns='jabber:client' xmlns:stream='http://etherx.jabber.org/streams' version='1.0'>
<?xml version='1.0'?>
<stream:stream xmlns='jabber:client' xmlns:stream='http://etherx.jabber.org/streams' from='jabber.org' id='5ce74cfce8e91fc4' version='1.0'>
  <stream:features>
    <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
    <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
      <mechanism>DIGEST-MD5</mechanism>
      <mechanism>PLAIN</mechanism>
    </mechanisms>
  </stream:features>
  <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
  <proceed xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
```

STARTTLS removal

Don’t negotiate whether to encrypt *over the network*!

- XMPP, IMAP, POP3, SMTP clients
- SMTP server-to-server *always works*
- Dell & Cisco BMCs that use the Avocent KVM stack (PoC||GTFO 0x5)
- HTTP (sslstrip by Moxie)

Forward Secrecy

Ephemeral Key Exchange is another form of public key cryptography

- Protocols: *Diffie-Hellman* or *Elliptic Curve Diffie-Hellman*
- Known as: *Forward Secrecy* or PFS
- TLS Ciphersuits that start with DHE- or ECDHE-
- The SSLv3 ciphersuites use RSA encryption - if the RSA key is stolen/cracked, past traffic can be decrypted! (Wireshark supports this)

Forward Secrecy - TLS Ciphersuites

Apache:

```
SSLCipherSuite ...
SSLHonorCipherOrder on
SSLProtocol all -SSLv2 -SSLv3
```

Nginx:

```
ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
ssl_prefer_server_ciphers on;
ssl_ciphers "...";
```

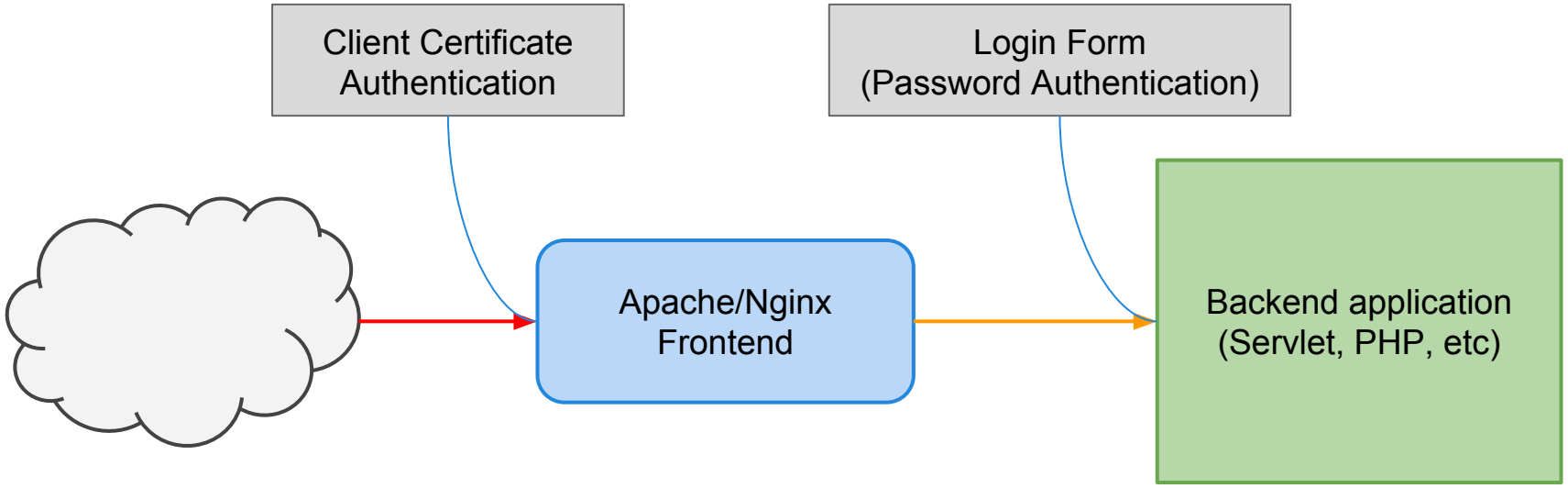
5 Ciphers that have you covered (thanks to Kenn White):

```
ECDHE-RSA-AES256-GCM-SHA384 # Android 4.4+
ECDHE-RSA-AES128-SHA256 # IE 11
ECDHE-RSA-AES128-SHA # Android 4.x, Chrome/Firefox, IE8-10
DHE-RSA-AES128-SHA # Android 2
RC4-SHA # Old junk (Windows XP, Nokia 6xxx) RC4 MUST DIE!
```

TLS - Authenticating Clients

Originally SSL was for e-commerce. This only required “money green” authenticity for clients.

TLS has support for client certificates



Entropy

Most cryptography needs randomness, for both short-term and long term keys.

The properties that are needed:

- Unable to predict future values
- Unable to recover past values

PRNGs work but need to be seeded from truly unguessable events.

Not Entropy

- mt_rand()
  - Can recover all state from output.
  - Often a small input
- rand() / random()
  - Small input
  - Can recover some/all state from output
- rand_r() / qrand() / java.util.Random
  - Small input
  - Small state
  - Can recover some state from output

https://www.miknet.net/rux2013/

Entropy - Don’t fork it up

- Unix-like systems: read from /dev/urandom
  - Userland PRNGs probably not fork() safe
- Windows: CryptGenRandom for strong entropy
- Linux early boot (only): /dev/random
  - Encrypted swap
  - SSH host key generation

Hash Functions - attack types

A fixed-length *digest* of variable length input

- (First)-Preimage resistance
  - Hard to find the original input from the hash
  - Guessing inputs still works!
- Second-Preimage resistance
  - Hard to find a second input that produces a given hash
  - An ideal hash function would provide $2^{\text{hash length}}$ resistance to this
- Collision Resistance
  - Hard to find two inputs that produce the same hash
  - Birthday attack - requires 256-bit hash for 128-bit security
  - When a hash function is broken this is usually first to go

Hash Functions - attack examples

- If the attack controls multiple inputs, you need to worry about *collisions*
  - rsync/librsync (see my github)
  - X.509 (TLS) certificates
- If the attacker controls one input, you need to worry about *second-preimage*

Checksums and Signatures

For general use, you should use SHA-2 256/384/512

Creating a certificate:

```
openssl req -new -sha256 …
```

Checksum of a file:

```
sha256sum *.iso
```

blake2 - is a very fast and secure hash function - if performance is critical.

Don’t use MD4/MD5 at all

SHA-1 should be phased out

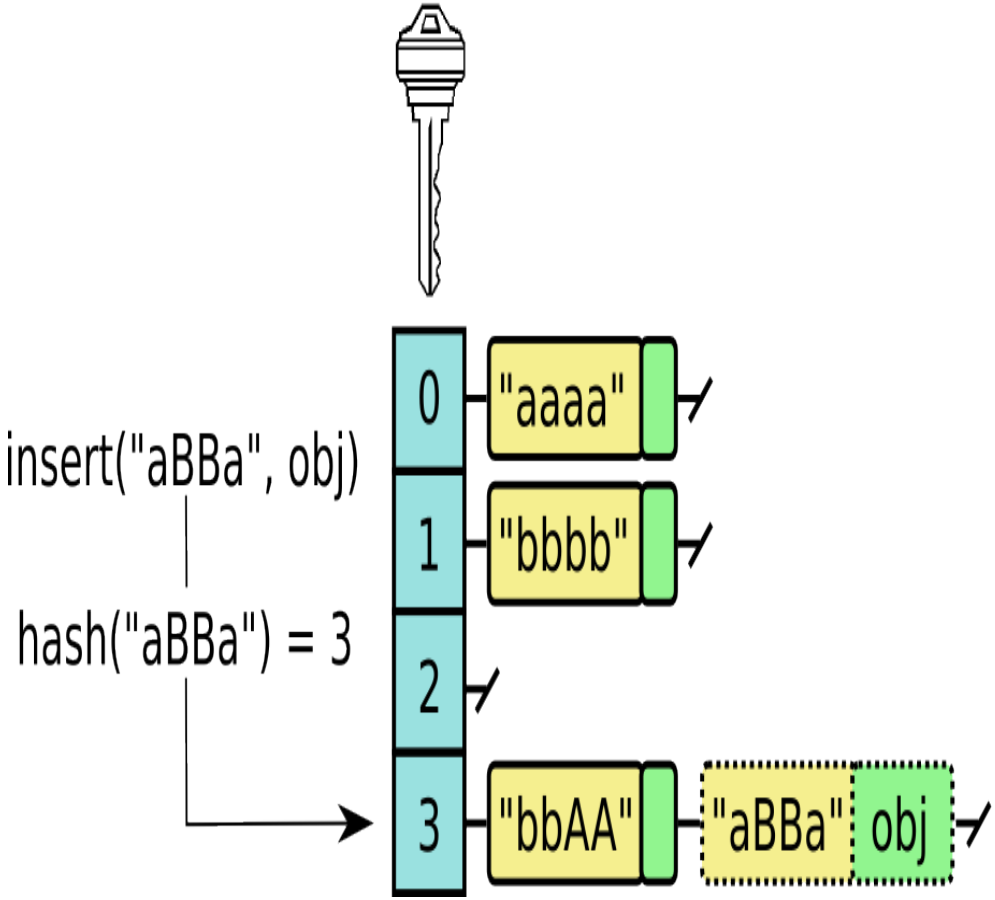
Hash Tables

Wait, do these need to be secure?!?

Worker pools or select() loops - colliding hash table entries can block the CPU!

SipHash was designed to fix this

This is now the default in Python3, Ruby, Perl



http://commons.wikimedia.org/wiki/File:Hashtable_linkedlist_collision.png

Password Hashes

Normal hash functions allow you to make extremely fast guesses - do not use these!

Salting

A salt is a unique string that is hashed with the password and stored next to the hash.

- ➢ Mutliple users with the same password won't have the same hash
- ➢ An attacker can't pre-calculate passwords

Stretching

- ➢ An operation that makes the hashing deliberately slow
- ➢ Must be sure that attackers can't take a shortcut

Current recommendation: bcrypt

Future recommendation: Winner of PHC - https://password-hashing.net/

Advanced Password Hashing - HSM

Passwords are still really weak!

Solar Designer: encrypt *hashes* with a HSM

- Near-perfect security if HSM is safe
- Better than nothing if HSM is stolen/broken
- Only requires *encrypt* function of HSM

Store:

```
salt, AES(bcrypt(password, salt))
```

Compare:

```
AES(bcrypt(password, salt)) == stored
```

http://www.openwall.com/presentations/YaC2012-Password-Hashing-At-Scale/

http://www.openwall.com/presentations/Passwords12-The-Future-Of-Hashing/

Password Hashing - Remediation

If you store your passwords in cleartext - go hash them all!

If you use an unsalted hash - use a password hash on the original hash

Avoid HTTP Digest Authentication, NTLM, CHAP.

MACs

If you need an untrusted entity to hold some state for you, you can use a MAC

- Ensure your data cannot be used out of context
  - HKDF, or just separate keys
- The key needs to be *secret* and preferably *random*
- Timing attacks! Brad Hill's trick:

```
HMAC(random, mac) == HMAC(random, HMAC(secret, data))
```

If the data needs to be encrypted MAC the ciphertext (EtM)

Hash functions aren't MACs!

Most hash functions do not function as a MAC.

Bad:

```
tag = Hash(secret, message)
```

and send `tag, message` that person can then perform a *length extension* attack.

Use HMAC when there's a secret key

Homework

- Write a MiTM attack for TLS and/or SSH
- Try it against every connection that leaves your machine
- File bug reports
- Code a length-extension attack
- cryptopals.com (if you like this stuff)

Thanks!

Comments/Questions?

Michael Samuel

Web	https://www.miknet.net/
Twitter	@mik235
GitHub	therealmik