Knock, Knock!
Who's There…

BSidesRDU 2022

Jason M. Pittman

…A New Kind of Port Knocking

Who am I…

Academic = {UMGC, HPU, Cal Poly…}

Technologist = {Big companies, startups, consulting}

Passions={internetworking, systems, beer}

THE INTRODUCTION

Is port knocking detectable, he asked.

Do you even network, bro?

…I know the OSI model, he exclaimed.

…well then, let’s find out, I retorted.

THE BACKGROUND

When was port knocking first announced?

Borss 2001

Kryzywinski 2003

de Graaf 2006

Does port knocking work in practice?

Well, yes

Simple concept

Authenticated connection vs. Connect -> Authenticate

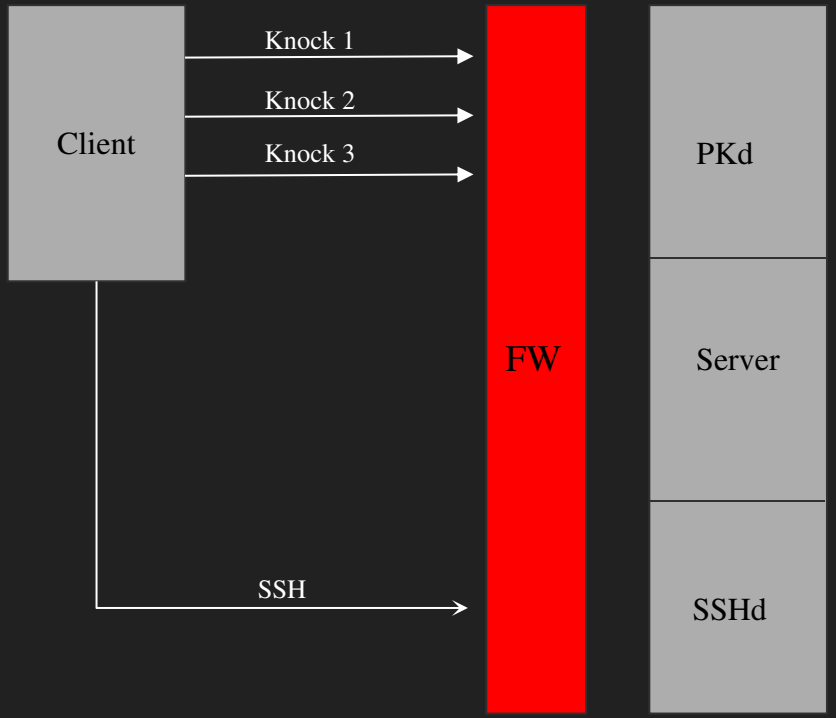
Core principles…

Concealment

Service protection

Authentication

What port knocking looks like…

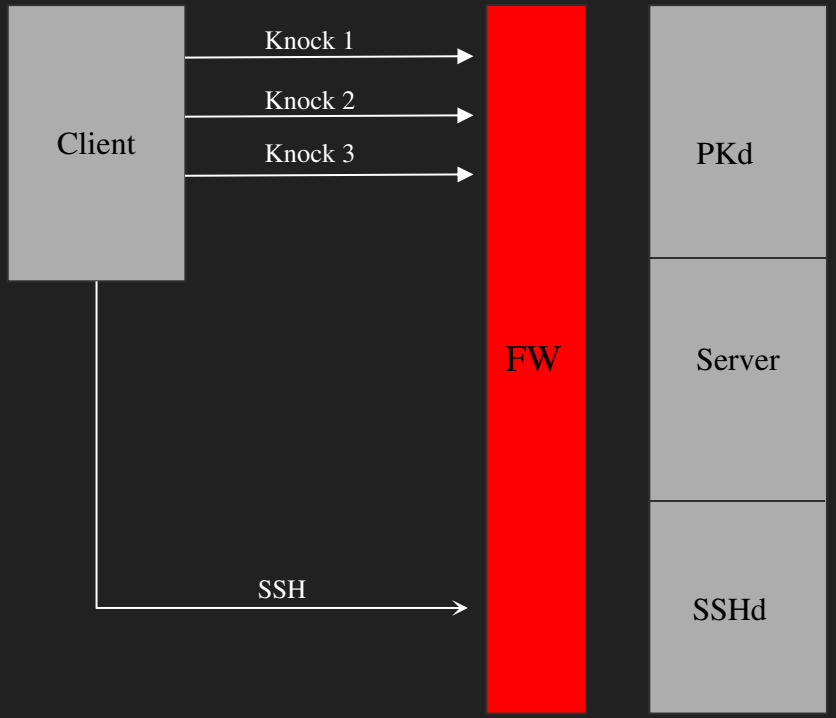


Is it detectable…?

Well, yes.

Two caveats

Are there systemic issues though…?



Specific *threats* to port knocking

Replay attacks

Spoofing

Anything else…?

…a trunk full of issues

Have there been any attempts to solve these problems?

Single packet authentication (SPA)

Sequence numbering schemes

Cryptography

…and now, Kommen!

THE DESIGN

Principles

Functional

Multi-user, pseudo anonymous

Resilient

Scalable & centralized management

What would a new port knocking algorithm look like?

Register clients

Preamble

Remote access sequence & codes

IPTables sorcery

THE IMPLEMENTATION

The core components…

kommen

kommen_server

kommen_service

The supporting cast…

Three crypto handlers

Eight utility handlers

What does pseudo-anonymous registration look like…?

Crypto keys

Fingerprint ID

Counter

```
def register_client(self, client_id=None):

    # generate keypair and get ids
    asym = asym_crypto.AsymmetricCryptographyHandler()
    sym = sym_crypto.SymmetricCryptographyHandler()

    # outer conditional to handle if client_id is none or
    if client_id is None:
        created = asym.create_keys()
    else:
        created = asym.create_keys(client_id)

    if created[0] is True:
        # build a new client object
        new_client = client.ClientHandler()
        new_client.client_name = created[1]
        new_client.client_id = created[1]
        new_client.client_status = 1
        new_client.client_pubkey = created[1]
        new_client.client_privkey = created[1]
        new_client.client_symkey = created[1]
        new_client.client_count = 1
```

The preamble…

Client initiated

Client id +

RACS +

Current Counter

Server internal check

ACK and send the RACS to IPTables

```python
def handle_premable(self, data):
    # this throws an error when the client ctrl-c disconnects...need to wrap for robustness
    payload = tuple(x for x in data.decode("utf-8").strip().split(','))

    #Server checks clients object for client_id:
    #if the client_id is in clients, server replies with a received_valid message
    is_valid = self.is_valid_preamble(payload[0], payload[1])

    #if the client_id is not in clients, server replies with a received_invalid message

    #if the payload is sanitized, we return true else false
    is_sanitized = self.sanitize_premable(payload)

    if is_valid and is_sanitized:
        print('True')
    else:
        print('False')

    #return plaintext as tuple(client_id, count, key)
    return payload
```

Remote access code sequence

15 digit HOTP (0*3 through 65535*3)

```
def generate_rac(self, counter):
    """Generate a remote access code
        Args:
            counter(int):

        Returns:
            rac(int):
    """
    try:
        rac = remote_access_code.HOTP(self.__secret, self.__length, digest=hashlib.sha512)
    except Exception as e:
        print('Error in generate_rac at line 57 as ' + str(e))

    return rac.at(counter)
```

IPTables core…

Default rules & for each unique preamble:

Build new RAC chains

SSH access

```
def __build_SSH_chain(self, table, c, ports):
    if not self.is_rac_chain_present('SSH_' + c):
        print('SSH chain not present...creating it now')
        try:
            #create the chain
            subprocess.run(["iptables -N SSH_" + c], shell=True)

            # clear the prior flag
            subprocess.run(["iptables -A SSH_" + c + " -m recent --name AUTH3_" + c + " --remove"], shell=True)

            # open ssh after racs
            subprocess.run(["iptables -A SSH_" + c + " -p tcp --dport 22 -j ACCEPT"], shell=True)

            # send traffic back to RAC1
            subprocess.run(["iptables -A SSH_" + c + " -j RAC1_" + c], shell=True)
        except Exception as e:
            print('Error from FirewallHandler in __build_SSH_chain as ' + str(e))
```

Our current status…

Beta

Known bugs

Upcoming features

github.com/Salus-Technology/kommen

Questions?