RedTeam presentation on Mock Banking System
Cryptography & Network Security (CSCI-4230)

Brian Sheedy & Avi Weinstock

December 9, 2015

Successful Attacks

- Denial of Service
  - Cause connection refusal (+ 100% CPU usage)
  - Kill bank process
- Man in the Middle Attack (Session Key Disclosure)
  - Eavesdrop
  - Steal everyone's money
  - Dispense infinite money

Connection Refusal DoS

- Bank does not handle disconnected ATMs properly (e.g. ˆC)
  - Causes the thread handling the connection to begin infinite loop
  - Thus doesn't close socket descriptor
- Repeatedly connect and disconnect ATMs
- Results in bank running out of socket descriptors and refusing further connections
- Also maxes out the CPU, causing host to become extremely slow

Process Killing DoS

- Two ways of killing bank
- Cause a SIGPIPE signal
  - Caused by trying to read from a closed descriptor
  - Achieved by repeatedly opening ATMs, logging in, logging out, and immediately killing the ATM process
- Send back fewer bytes than expected in the key exchange
  - CryptoPP expects exactly 384 bytes, throws an exception if input differs
  - Input not checked before handing to function
  - Achieved by having the proxy send back an arbitrary string that's shorter than 384 bytes

Key Exchange DoS Demo

Man in the Middle Attack

- ATM sends the session key to the bank after receiving the bank's public RSA key
- However, ATM does not know whether the public key it receives is actually the bank's
- Can intercept the bank's public key and send our own public key to the ATM
- We receive the encrypted session key, decrypt it, encrypt it with the bank's public key, and send it to the bank
- Now we know the AES session key
  - Can passively eavesdrop and steal PINs
  - Can modify any passed messages
  - Can imitate the bank

Specific Man in the Middle Attack Examples

- Steal everyone's money
  - Anytime someone attempts to log in to the bank, log in before them and transfer all their money to Eve (and then log them in normally)
- Dispense infinite money
  - Log in to an ATM
  - Make a withdrawal request
  - Intercept message to bank and reply with a message approving the withdrawal
  - ATM dispenses the money without any money being deducted from the account

Man in the Middle Attack Demo(s)

RCE Attempt

- During the key exchange, the AES Key and IV are decrypted with RSA-OAEP-SHA1
- Their lengths aren't checked, and the maximum payload size is 342
- Only enough space is allocated for 16 byte keys/nonces
- Sadly, this isn't obviously exploitable because there's a socket descriptor that acts as a canary
- (`read` returns `EBADF` in an infinite loop)
- If errors were actually handled correctly, this would be trivially exploitable

RCE Attempt (Vulnerable code)

```
// Read encrypted AES key
int num_read = read(client_fd, buf, HANDSHAKE_BUFFER_SIZE);

std::string cipher(buf, num_read), recovered;
CryptoPP::RSAES_OAEP_SHA_Decryptor d(privateKey);
CryptoPP::StringSource ss1(cipher, true,
    new CryptoPP::PK_DecryptorFilter(rng, d,
        new CryptoPP::StringSink(recovered)
    )
);

sprintf(buf, "DUMMY");
write(client_fd, buf, 5); // Dummy write to finish proxy transaction

sprintf(reinterpret_cast<char*>(aes_key), "%s", recovered.c_str());

char tmp[HANDSHAKE_BUFFER_SIZE];
strcpy(tmp, reinterpret_cast<char*>(aes_key));

// Read the encrypted initialization vector
char iv_buf[HANDSHAKE_BUFFER_SIZE];
int iv_num_read = read(client_fd, iv_buf, HANDSHAKE_BUFFER_SIZE);

std::string cipher2(iv_buf, iv_num_read);
std::string recovered_iv;
CryptoPP::RSAES_OAEP_SHA_Decryptor d2(privateKey);
CryptoPP::StringSource ss2(cipher2, true,
    new CryptoPP::PK_DecryptorFilter(rng, d2,
        new CryptoPP::StringSink(recovered_iv)
    )
);

sprintf(reinterpret_cast<char*>(iv), "%s", recovered_iv.c_str());
strcpy(reinterpret_cast<char*>(aes_key), tmp);
```

RCE Attempt (Stack diagram)

bank_aes_handshake arguments

EBP + 8	int client_fd
EBP + c	PrivateKey &privateKey
EBP + 10	PublicKey &publicKey
EBP + 14	byte* aes_key
EBP + 18	byte* iv
EBP + 1c	std::string& init_nonce

main locals

EBP - 29c	int client_sock
EBP - 268	char port_str[10]
EBP - 238	client_info client_args
EBP - 238	client_args.sockfd
EBP - 234	client_args.privateKey
EBP - 230	client_args.publicKey
EBP - 22c	pthread_t client_thread
EBP - 224	socklen_t addr_size
EBP - 220	struct sockaddr_storage client
EBP - 190	int listen_sock
EBP - 18c	struct addrinfo *res
EBP - 188	struct addrinfo hints
EBP - 164	pthread_t console_thread
EBP - 160	PublicKey publicKey
EBP - 128	PrivateKey privateKey
EBP - 28	std::string inputPort

thread_handle locals

EBP - 1cc	std::string nonce
EBP - 1c0	action::Action response
EBP - 100	action::Action action
EBP - e0	std::string s
EBP - 48	byte iv[16]
EBP - 38	int dummy_alloc
EBP - 34	byte aes_key[16]
EBP - 24	PublicKey *publicKey
EBP - 20	PrivateKey *privateKey
EBP - 1c	int client_sock
EBP - 18	client_info *client_args