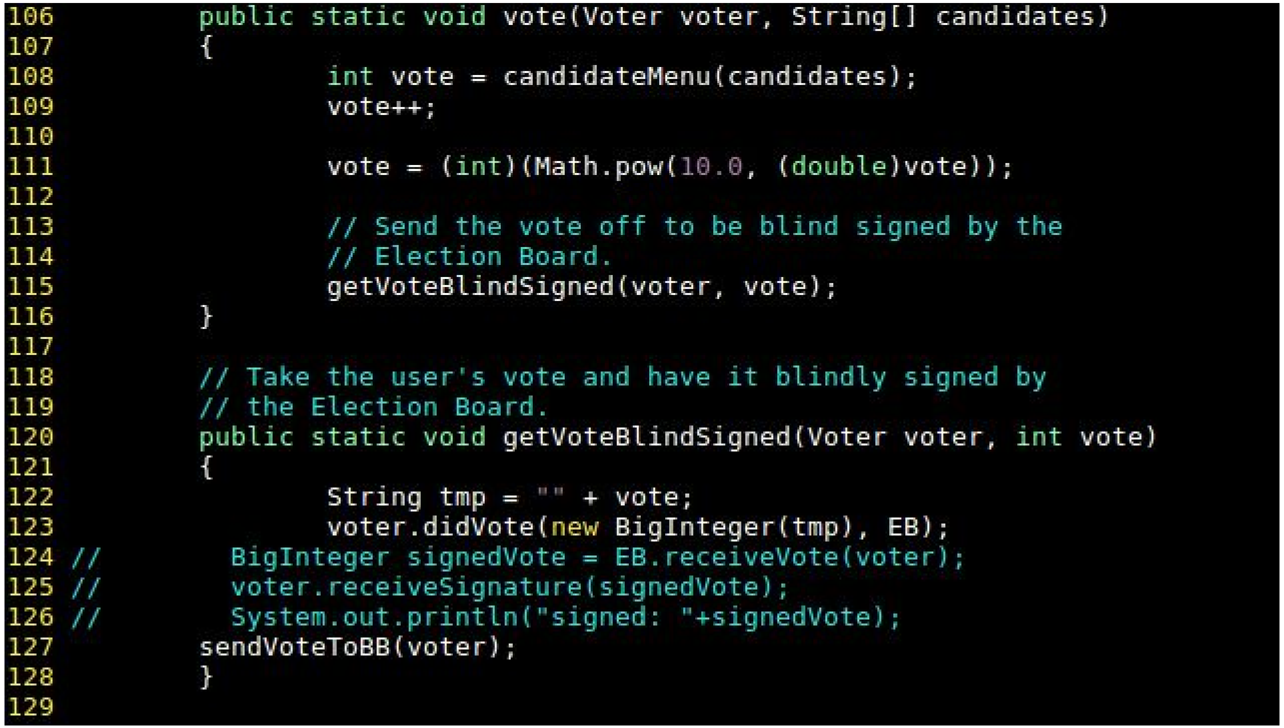
Red Team

For Groups 17, 23

17 - Privacy Issue - EVoting.java

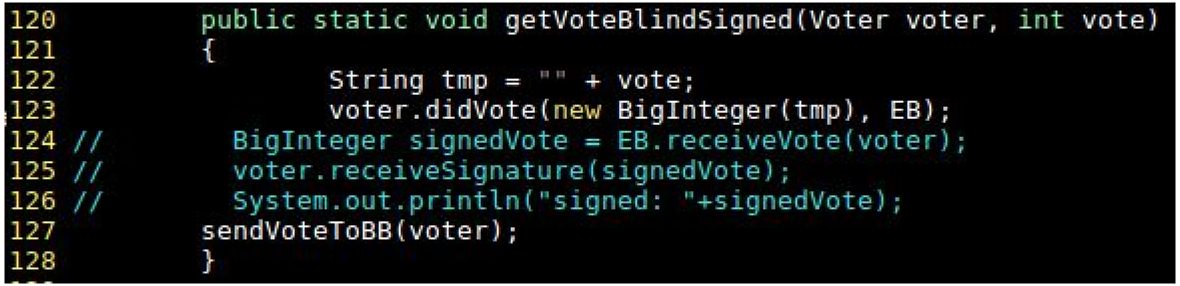


```java
	public static void vote(Voter voter, String[] candidates)
	{
		int vote = candidateMenu(candidates);
		vote++;

		vote = (int)(Math.pow(10.0, (double)vote));

		// Send the vote off to be blind signed by the
		// Election Board.
		getVoteBlindSigned(voter, vote);
	}

	// Take the user's vote and have it blindly signed by
	// the Election Board.
	public static void getVoteBlindSigned(Voter voter, int vote)
	{
		String tmp = "" + vote;
		voter.didVote(new BigInteger(tmp), EB);
//		  BigInteger signedVote = EB.receiveVote(voter);
//		  voter.receiveSignature(signedVote);
//		  System.out.println("signed: "+signedVote);
	sendVoteToBB(voter);
	}

```

- Takes in user’s vote
- getVoteBlindSigned()

17 - Privacy Issue - EVoting.java

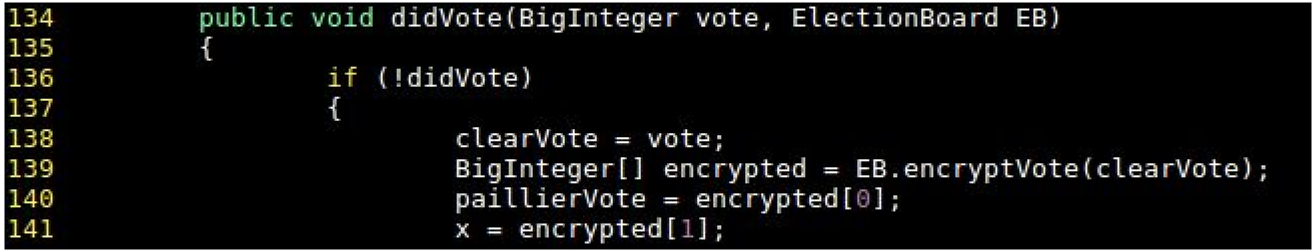
- tmp = “” + vote;
- voter.didVote(tmp, EB);



```java
	public static void getVoteBlindSigned(Voter voter, int vote)
	{
		String tmp = "" + vote;
		voter.didVote(new BigInteger(tmp), EB);
//		BigInteger signedVote = EB.receiveVote(voter);
//		voter.receiveSignature(signedVote);
//		System.out.println("signed: "+signedVote);
	sendVoteToBB(voter);
	}
```

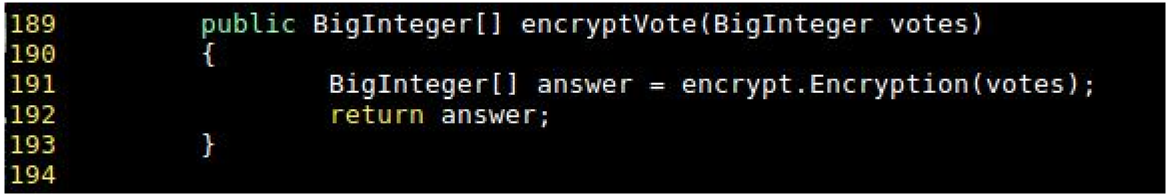
17 - Privacy Issue - Voter.java

- Saves vote as clearvote
- EB.encryptVote(clearVote)
  - Vote sent to EB



```java
	public void didVote(BigInteger vote, ElectionBoard EB)
	{
		if (!didVote)
		{
			clearVote = vote;
			BigInteger[] encrypted = EB.encryptVote(clearVote);
			paillierVote = encrypted[0];
			x = encrypted[1];
```

17 - Privacy Issue - ElectionBoard.java

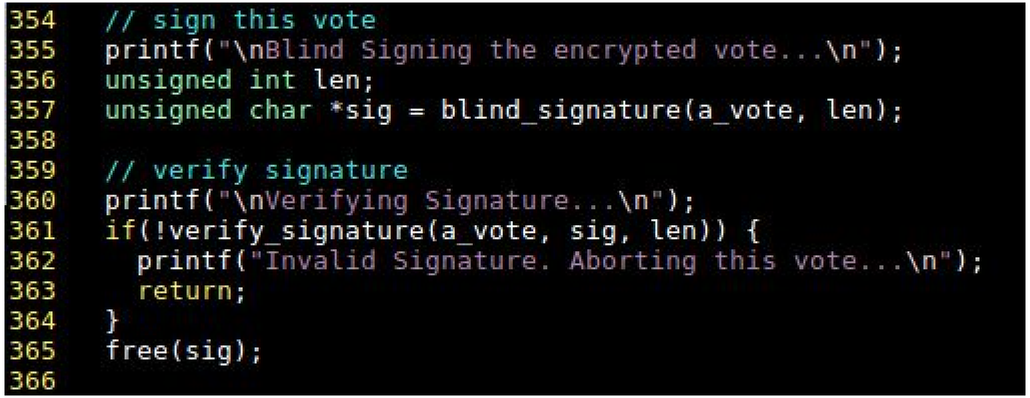


```
        public BigInteger[] encryptVote(BigInteger votes)
        {
                BigInteger[] answer = encrypt.Encryption(votes);
                return answer;
        }

```

- The clearVote is sent as a parameter variable.
    - Thus, the EB sees the unencrypted & unblinded vote.
    - Vote is also on the stack.

23 - Bundled

- Current implementation has one main method play the role of the EB, Voter, and BB.
- Creates multiple vulnerabilities.
    - E.g. Vote is blind signed and verified by the same entity. So, Voter can sign and send any other vote.

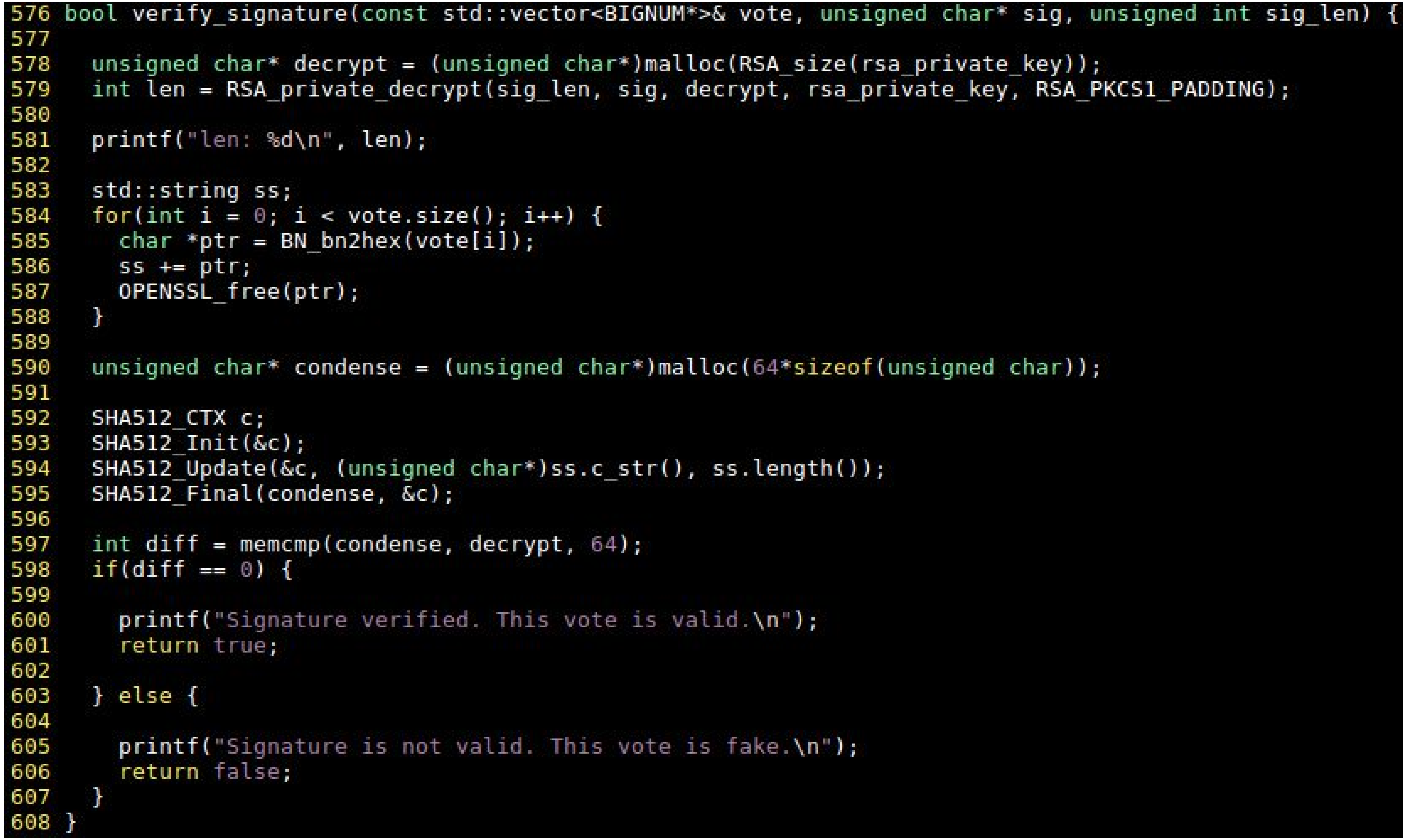


```
  // sign this vote
  printf("\nBlind Signing the encrypted vote...\n");
  unsigned int len;
  unsigned char *sig = blind_signature(a_vote, len);

  // verify signature
  printf("\nVerifying Signature...\n");
  if(!verify_signature(a_vote, sig, len)) {
    printf("Invalid Signature. Aborting this vote...\n");
    return;
  }
  free(sig);

```

23 - I don’t want to vote

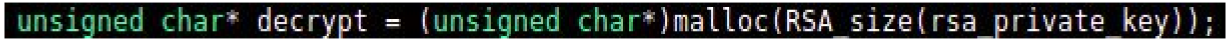
- If one voter exits, whole system stops, meaning all votes are lost.

23 - Memory Leak

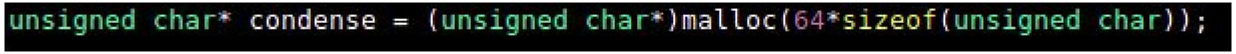


```cpp
bool verify_signature(const std::vector<BIGNUM*>& vote, unsigned char* sig, unsigned int sig_len) {

  unsigned char* decrypt = (unsigned char*)malloc(RSA_size(rsa_private_key));
  int len = RSA_private_decrypt(sig_len, sig, decrypt, rsa_private_key, RSA_PKCS1_PADDING);

  printf("len: %d\n", len);

  std::string ss;
  for(int i = 0; i < vote.size(); i++) {
    char *ptr = BN_bn2hex(vote[i]);
    ss += ptr;
    OPENSSL_free(ptr);
  }

  unsigned char* condense = (unsigned char*)malloc(64*sizeof(unsigned char));

  SHA512_CTX c;
  SHA512_Init(&c);
  SHA512_Update(&c, (unsigned char*)ss.c_str(), ss.length());
  SHA512_Final(condense, &c);

  int diff = memcmp(condense, decrypt, 64);
  if(diff == 0) {

    printf("Signature verified. This vote is valid.\n");
    return true;

  } else {

    printf("Signature is not valid. This vote is fake.\n");
    return false;
  }
}
```

23 - Memory Leak



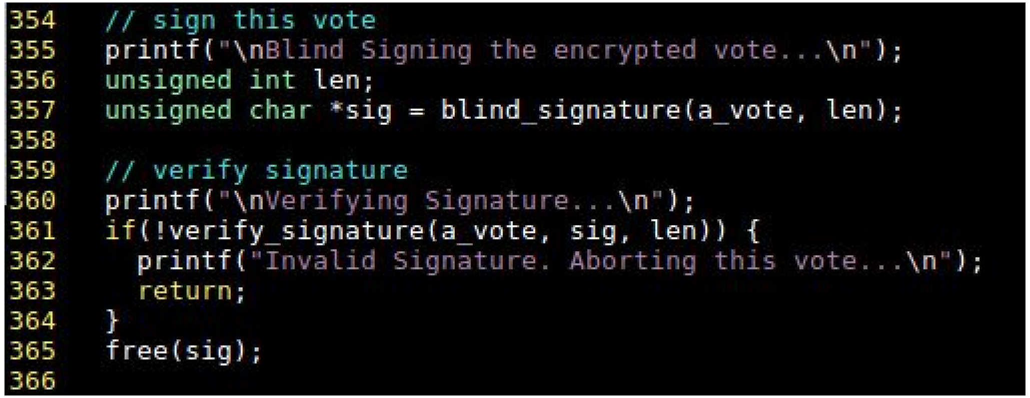
```
unsigned char* decrypt = (unsigned char*)malloc(RSA_size(rsa_private_key));
```



```
unsigned char* condense = (unsigned char*)malloc(64*sizeof(unsigned char));
```

- Assuming RSA key is 256 bytes, every verify_signature() call takes 320 bytes.
- Can accumulate 1 TB of unfreed memory in about 3,000,000,000 method calls.

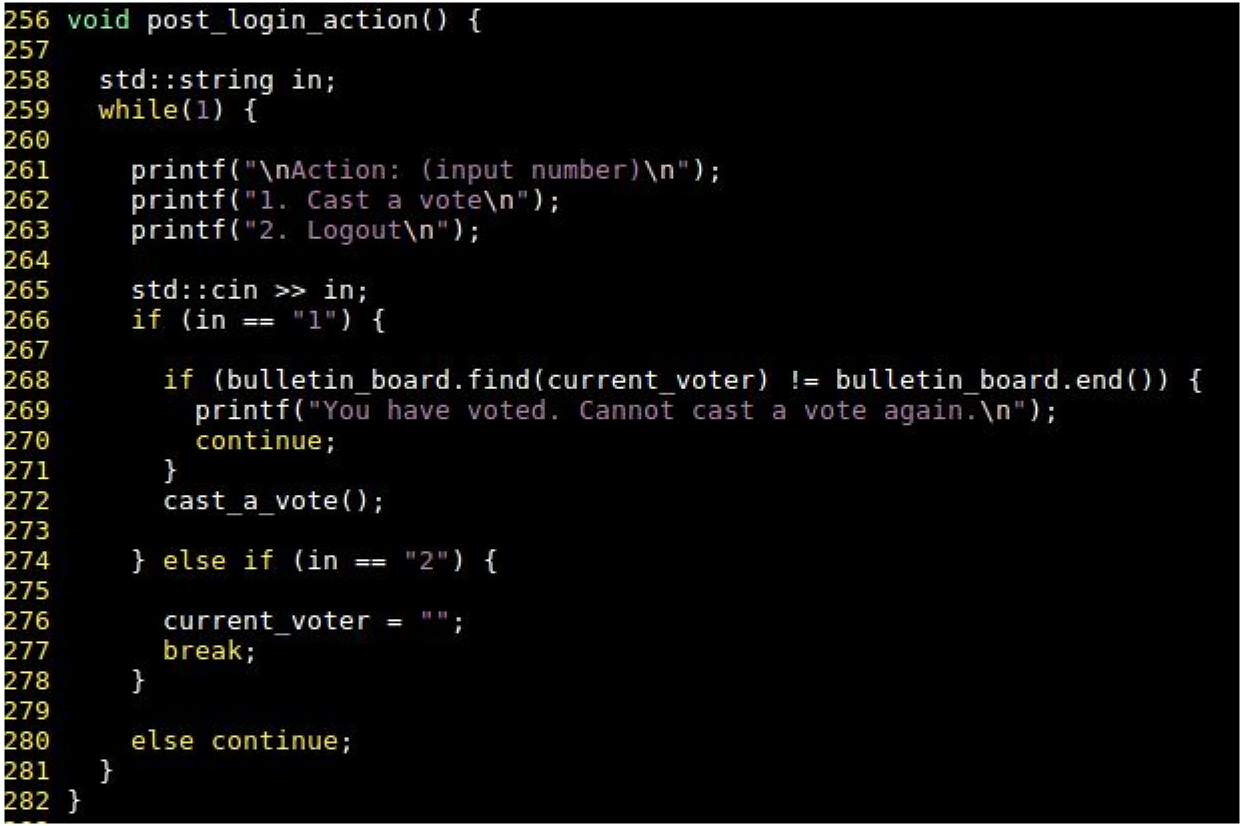
23 - Memory Leak



```
// sign this vote
printf("\nBlind Signing the encrypted vote...\n");
unsigned int len;
unsigned char *sig = blind_signature(a_vote, len);

// verify signature
printf("\nVerifying Signature...\n");
if(!verify_signature(a_vote, sig, len)) {
  printf("Invalid Signature. Aborting this vote...\n");
  return;
}
free(sig);

```

- This is from cast_a_vote()
- Invalid signature does a return to …

23 - Memory Leak



```
void post_login_action() {

  std::string in;
  while(1) {

    printf("\nAction: (input number)\n");
    printf("1. Cast a vote\n");
    printf("2. Logout\n");

    std::cin >> in;
    if (in == "1") {

      if (bulletin_board.find(current_voter) != bulletin_board.end()) {
        printf("You have voted. Cannot cast a vote again.\n");
        continue;
      }
      cast_a_vote();

    } else if (in == "2") {

      current_voter = "";
      break;
    }

    else continue;
  }
}
```

- Calls cast_a_vote()
- Invalid signature just causes a return.
- Loop continues.
- Can repeatedly send incorrect signature pairs.

23 - Memory Leak

- Input parsing
  - 1 2 1 2 1 2 1 2… will use 1 as the option, then 2 for the next option, then 1 for ...
- After logging in, input “1 2” x 3,000,000,000
  - Choose to vote, vote for candidate 2 (vote gets aborted), choose to vote, ...
- Produced string in a second.
- Printed string in 10 seconds.