Computer Security Seminar
API Attacks

Security Engineering/Ross Andersson, Chapter 18

Shai Ziv

25th May, 2014

Application Programming Interface

- Interface for communication between two programs.
  - Two threads of the same program.
  - Two programs running on the same server.
  - Client and server.

API is Vulnerable

- Door to the outer world.
- Untrusted sources give commands.

- Designing a secure API is very difficult.

- Small programming oversights can be disastrous.

The Perfect API

```
void API(void)
{
        printf("No commands available");
}
```

Useful, eh?

Attack on Visa Security Module

- Hardware device for bank security.
- Stores no memory.
  - Only a single master key stored in tamper-resistant memory.
  - Encryption under this key is “unbreakable”.

- We will look at the Terminal Key Generation for ATMs
- ATM security is based on dual control (secret sharing)
  - $K = K_1 \oplus K_2$

Attack on Visa Security Module – cont.

- Key creation:

$$Program \xleftarrow{E_M(K_1)} \boxed{VSM} \xrightarrow{K_1} Worker\ 1$$

$$Program \xleftarrow{E_M(K_2)} \boxed{VSM} \xrightarrow{K_2} Worker\ 2$$

$$Program \xrightarrow{E_M(K_1), E_M(K_2)} \boxed{VSM}$$

$$Program \xleftarrow{E_M(K = K_1 \oplus K_2)} \boxed{VSM}$$

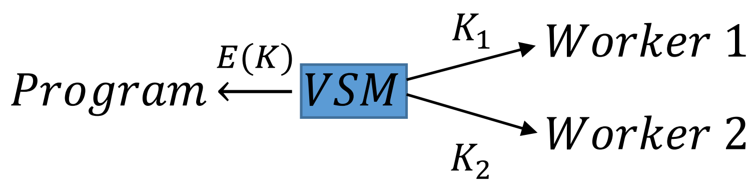
- What happens if we insert the same encrypted key twice?
  - $K = K_1 \oplus K_1 = 0.$
  - Known key inside the system.

Attack on Visa Security Module – cont.

- The problem: Support of offline ATMs.
  - $Program \xrightarrow{E_M(PIN),E_M(K)} \boxed{VSM}$
  - $Program \xleftarrow{E_K(PIN)} \boxed{VSM}$
- $PIN = D_0(\cdot)$.

Attack on Visa Security Module – cont.

- How to fix?
- Independent atomic commands!



Attack on IBM PIN Generation

- Wild credit cards appear!
- IBM uses PIN generation.
- It's not very effective…

- In IBM PIN code generation, PIN_C depends on 3 values:
  - PIN_M – bank's master PIN.
  - N_C – account number.
  - $offset$ – for memorable (weak) PIN.

Attack on IBM PIN Generation – cont.

- The algorithm:

$$Hex = E_{PIN_M}(N_C) \qquad Hex = a2ce126c69aec82d$$
$$Dec = Dec\_Table(Hex) \qquad Dec = 0224126269048 23$$
$$PIN_C = Dec[1..4] + offset \qquad PIN_C = 0224 + 6565 = 6789$$

- Great Idea (?): Dec_Table is supplied by the user.
  - $Dec\_Table = 0123456789012345$ was widely used.

Attack on IBM PIN Generation – cont.

- Set $Dec\_Table = 0000000000000000.$
  - Get $\mathrm{E}(PIN_C = 0000).$
- Set $Dec\_Table = 1000000000000000.$
  - If $E(PIN_C)$ changed, then it contained a '0'.
- And so on…
- With a few dozen queries PIN_C can be found.

Attack on IBM PIN Generation – cont.

- How to fix?
  - IBM's "solution":
    - Must contain at least 8 different characters, that appear at most 4 times.
    - What about "0123456789012345", then "1123456789012345", and so on?
- Be careful when using user's input, and avoid it as much as possible.
  - Remember the perfect API!

API Programming - Input Check

- The API itself can be 100% safe.
- The communication still will not be secure.

- Before you execute, check the input you are executing!

SQL Injection (Input Check – example)

- Many APIs use SQL transactions in the background.
- The code is written in advance,

  and the parameters are taken from the API call.
- If the parameter isn’t checked, SQL code can be ‘Injected’ and executed.

- SQL code:

$$\begin{aligned}&select\ *\\&from\ workers\\&where\ name = ('\$\$');\end{aligned}$$

- Expected parameter *($$)*: $Shai\ Ziv$

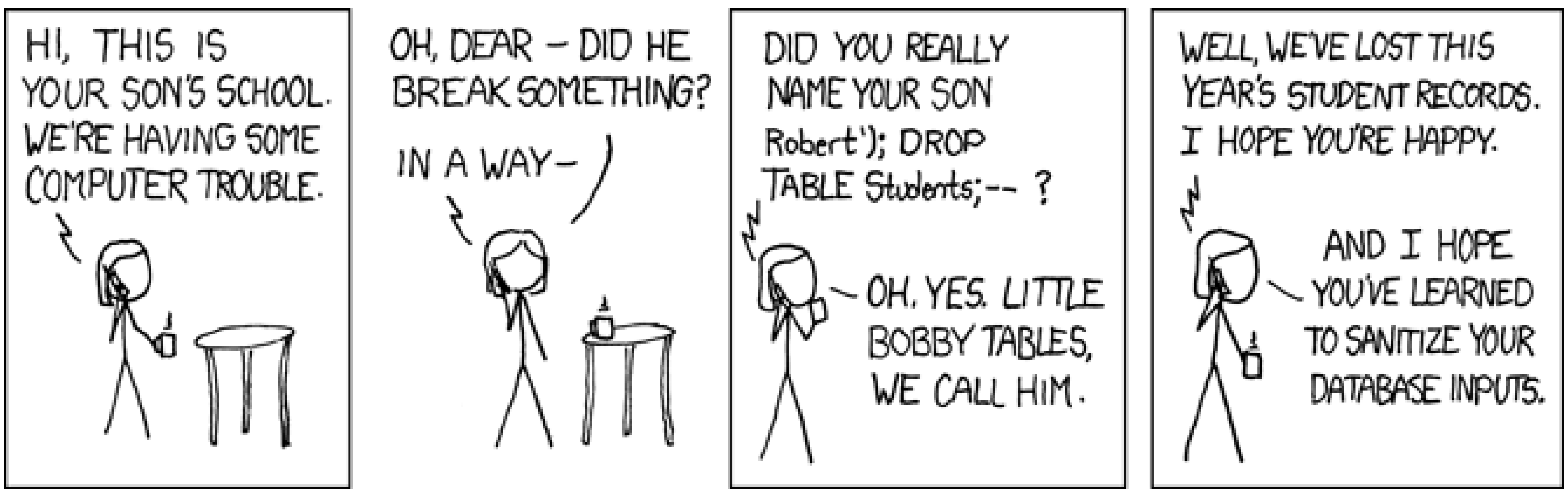
- Attacker's parameter:

$Shai\ Ziv'); \ insert\ into\ workers\ values\ ('Joffrey\ Baratheon$

- When inserted:

$$\begin{aligned}&select\ *\\&from\ workers\\&where\ name = ('Shai\ Ziv');\\&insert\ into\ workers\\&values\ ('Joffrey\ Baratheon');\end{aligned}$$

SQL Injection – cont.



Buffer Overflow (Input Check – example)

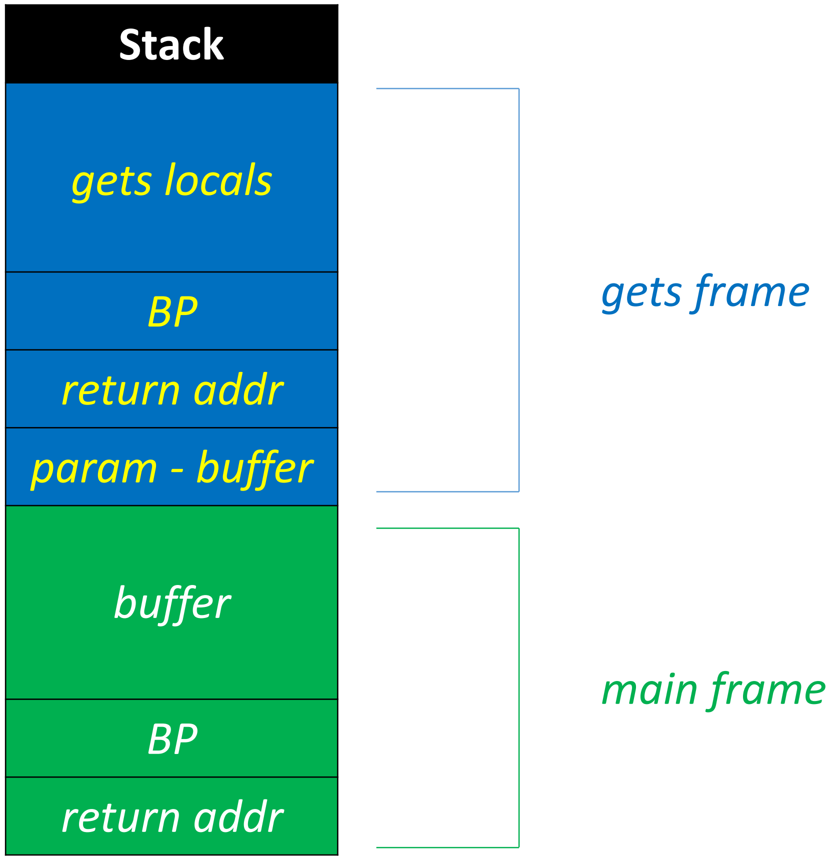
- Every API reads input from the user.
- No computer has an infinite input buffer.

- Devastating attacks can be executed if input string length is not checked.

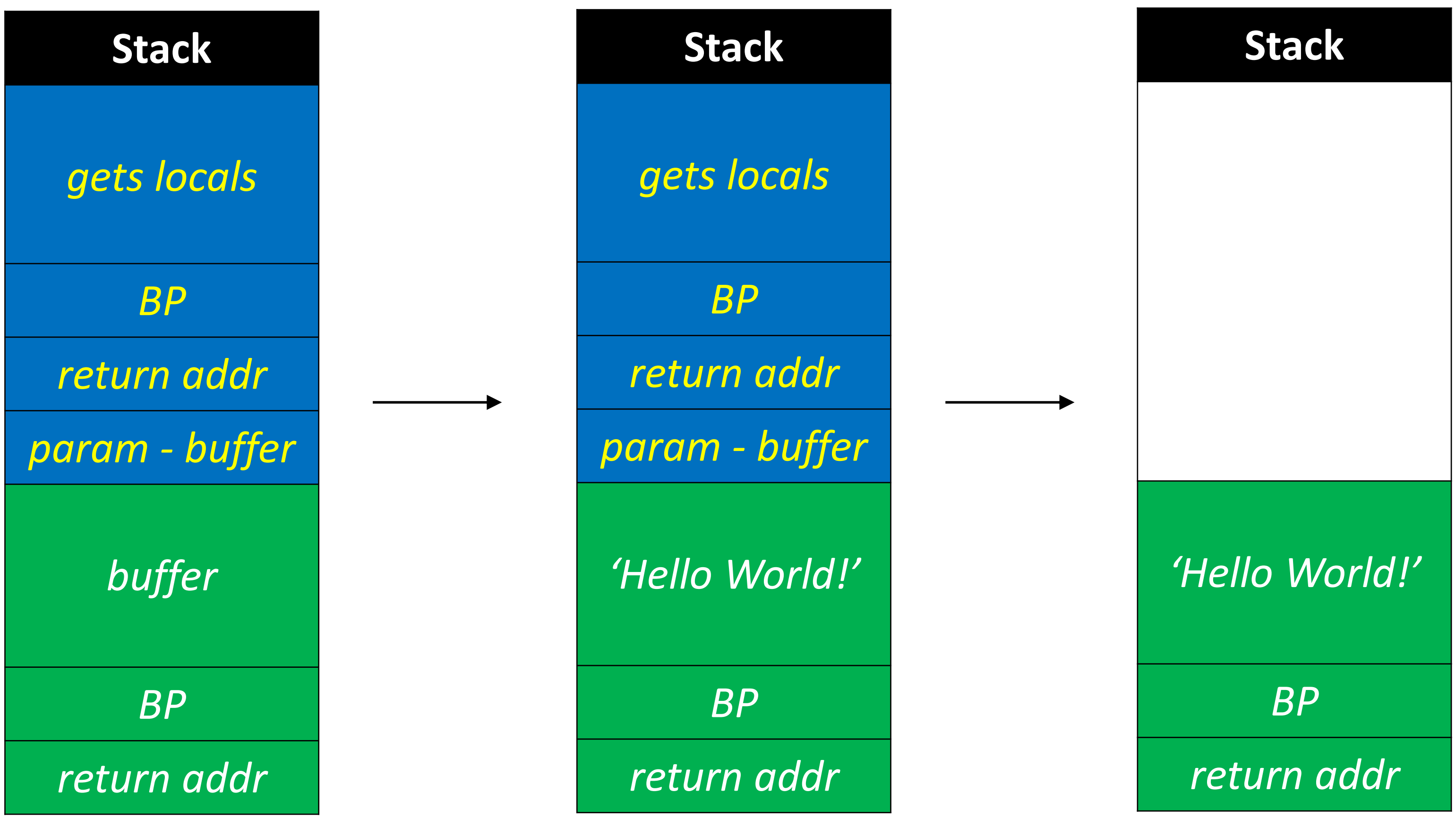
- What is the problem here?

$$main(\cdot)$$
$$\{$$
$$char\ buffer[128];$$
$$gets(buffer);$$
$$\}$$

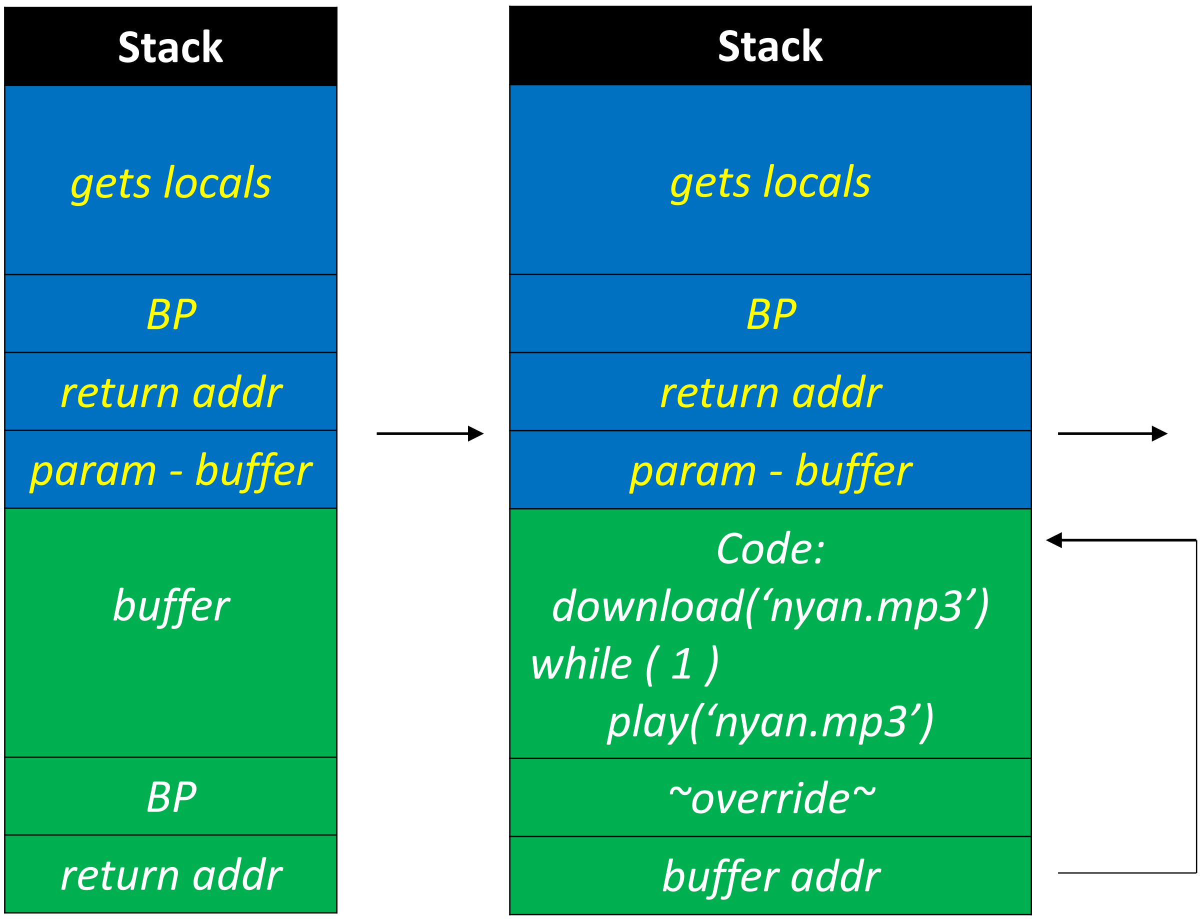
Buffer Overflow – The Stack



No Buffer Overflow – No Attack



Buffer Overflow – The Attack



- main finishes as usual
- Computer is infected

Summary – API Design

- Designing a secure set of commands is very difficult.

- Single secure looking command might be insecure.
- Multiple secure commands might be insecure when combined.
- Each user input can be used for an attack.

Summary – API Design – cont.

- Simplicity is key.
  - Complicated APIs are all the more vulnerable.
  - Atomic and independent commands.

- Many failures happen when adding features to API.
  - When designed initially, those features were not considered.
  - The feature itself should be checked and rechecked.
  - Relations between the new feature and old features might be problematic.
  - Is the feature necessary?

- Use as minimal input as possible.
  - There is no reason to use a parameter from the user,
  when you know its value in advance.

Summary – API Implementation

- Input check.
- Input check.
- Input check.

- The code which handles the user's input is extremely critical, and should be treated that way.

Backup

Attack on the 4758

- 4758 is IBM's equivalent to Visa's module.
- The 4758 supported "check value" creation for a key K
  - $check = E_K(0)$
- At the time the key length was 56 bits.
  - This means we need 2^{55} effort to crack an unknown key, which is (not really) too much.

Attack on the 4758 – cont.

- We do not need to crack a specific key.
  - The 4758 would re-encrypt data with a different key

- Meet in the middle attack:
  1. Collect a number of check values. Say 2^{16} (takes a few hours)
  2. Store them in a hash table.
  3. Go over keys until you get a hit. (takes $\frac{2^{56}}{2^{16}} = 2^{40}$ effort)
  4. ?????
  5. Profit.