#### **Purpose**

In this discussion, I would like you to explore certain web-­centric vulnerabilities to enhance your understanding of computer security, in particular web-­centric application/system security. By searching the Internet, please find one you like, which can be any disclosed/exploited (web-­centric application/system) vulnerability that led to a data breach and/or compromised systems.

#### **Tasks**

To make the discussion concise, please focus your discussion on **only one** disclosed/exploited vulnerability published on the Internet. Please submit one original post using the following guideline:

* Describe your finding/example.
* Describe the consequences due to the disclosed/exploited vulnerability.

If known, discuss how this given vulnerability can (or might) be removed.

The most interesting type of web-centric vulnerability to me is some type of SQL injection. This is because it seems so simple to avoid or protect against, but sometimes it can slip through the cracks and have serious consequences. A SQL injection occurs when a request to the backend is sent with SQL code written by the user. For example, if you logged into a site as a specific user and your user IDis in the URL as a parameter, i.e “?userId=999”, the user can modify the URL parameter to be “?userId=999 or 1=1”. The SQL piece is the “or 1=1”, which would always evaluate to True on the backend. Data for all users can potentially be returned instead of just your user.

The consequence of this is that this data may include email addresses, passwords, PII, or other protected data. This type of attack has affected huge companies, like Target and LinkedIn. Malicious actors were able to collect millions of records of personal user data. This can be a huge financial cost to the company if they have to pay to protect users or potentially fines if they have violated any laws/regulations. A consequence that would affect a user directly is if the bad actors gained access to credit card information. It can be used by the hackers to steal money directly from the affected user.

This type of vulnerability can be prevented with something as simple as input validation. If the expected input is a number, then the system should make sure that the entered value evaluates to a number. Or, if the input is string, maybe there is a character limit, or no whitespaces. Another way to prevent this is to use prepared scripts or commands, rather than concatenating user input. This will allow the system to know exactly what is being executed. In one of projects at work, we use a backend proxy to filter requests and responses. This way we can sanitize any input before it actually hits our databases.