Templates for Constructing the Attack Trees

May 29, 2017

This document presents the different templates that are used to automatically generate attack trees for the risk assessment of CPS. Dashed leaves are leaves of the final attack tree. Other leaves will be refined further by additional templates.

Modify a parameter.
The tree built by template *Modify(Parameter)* is shown in Figure 1. The following elements from the system model are taken into account:

- Which software modules m, m', . . . are able to control parameter p.
- Which networks n these software modules are located in.
- Which hardware components c, c', . . . are sensors that measure parameter p.

How many children the root node has depends on the system architecture. For every software module that can modify the parameter, a child node is added. These nodes are further matched with other templates which will extend the tree. Likewise, for every sensor that measures the parameter a leaf is added that will check an attacker's access set "A". Finally, the attacker could attempt a spoofing attack. For this route to succeed, the attacker must first gain access to a network from where he can access the modules, this node is matched with another template. The attacker must also be able to run spoofing software, to this end a leaf is added which checks the attacker's capability.

Obtain data assets.
The tree built by template *Obtain(Asset)* is shown in Figure 2. The following elements from the system model are taken into account:

- Which components c, c', . . . the data asset d is stored in.

If the attacker wants to obtain a data asset, this template checks in which components this data asset is stored. For each such component a node is added to the tree. This node is further refined in the next template.

Obtain a data asset from a component.
The tree built by template *Obtain(Asset, Component)* is shown in Figure 3.

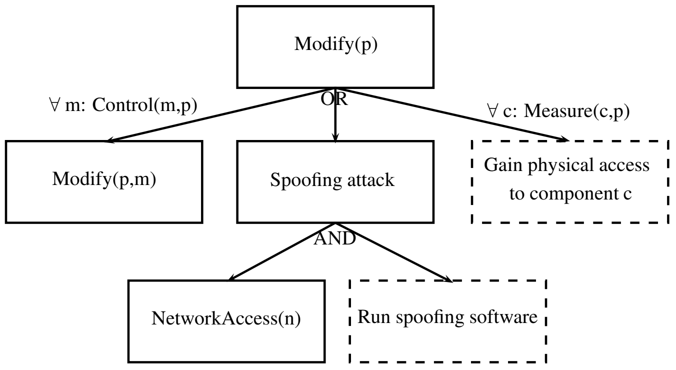

Figure 1: Template Modify(p).

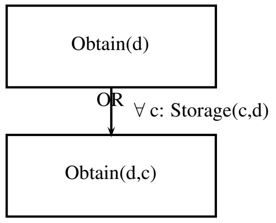

Figure 2: Template Obtain(d).

If the attacker wants to obtain a data asset from a specific component *c*, the attacker must first gain access to this component. This will be resolved by the *Access* template defined in Figure 8. Next, the attacker must exploit a data leakage vulnerability to obtain the data, this is handled by the next template.

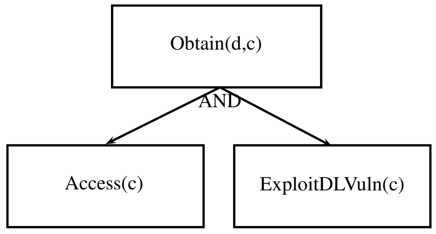

Figure 3: Template Obtain(d,c).

Exploit a vulnerability.
The tree built by template *ExploitDLVuln(Component)* is shown in Figure 4.

This template deals with exploiting data leakage vulnerabilities. Similar templates exist for the other vulnerability categories. The following elements from the system model are taken into account:

- Whether component c contains a known vulnerability of category Data Leakage.

If the component contains a known vulnerability, the attacker can attempt to exploit it. Otherwise, the attacker must first discover a vulnerability. This is handled by the next template.

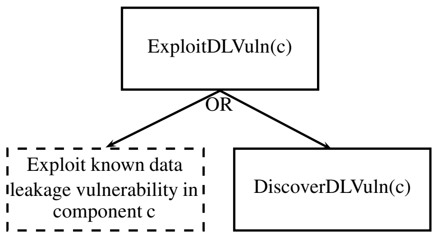

Figure 4: Template ExploitDLVuln(c).

Discover a vulnerability.
The tree built by template *DiscoverDLVuln(Component)* is shown in Figure 5. This template deals with discovering data leakage vulnerabilities. Similar templates exist for the other vulnerability categories. Here two leaves related to attacker capabilities will be added. First the attacker must discover a data leakage vulnerability, then he must exploit it.

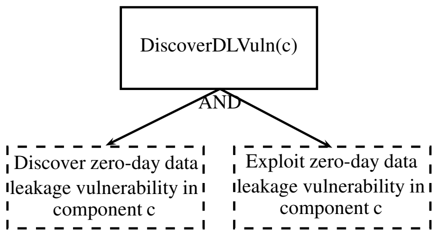
DiscoverDLVuln(c)
AND
Discover zero-day data leakage vulnerability in component c
Exploit zero-day data leakage vulnerability in component c

Figure 5: Template DiscoverDLVuln(c).

Halt the workings of the CPS process by performing a denial of service attack on a key system element.
The tree built by template *DoS(SystemPart)* is shown in Figure 6. To launch a DoS attack on a component or module, the attacker must first be able to access it. This will be resolved by the *Access* template defined in Figure 8. Next, the attacker must exploit a vulnerability to launch the DoS attack, this is handled

by templates similar to the ones in Figures 4 and 5.

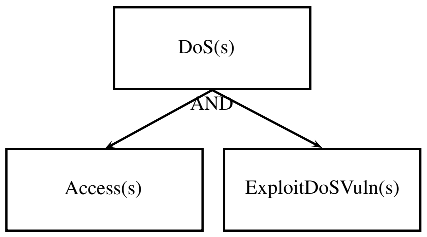

Figure 6: Template DoS(s).

Modify a process parameter using a software module.
The tree built by template *Modify(Parameter, Module)* is shown in Figure 7. The following elements from the system model are taken into account:

- Which component c contains module m.

When using a module to modify a parameter, the attacker has two options. He can attempt to use the module as a regular employee would. To this end, the attacker must have access to the component c containing m, he must be able to authenticate to m and he must be authorized to modify p from m. These three conditions correspond to templates shown in Figures 8, 10 and 13. Alternatively, the attacker could attempt to abuse a remote code execution vulnerability in the module m. The attacker must first gain access to the component which holds the module. Then the attacker must launch an attack. This is handled by templates similar to the ones in Figures 4 and 5.

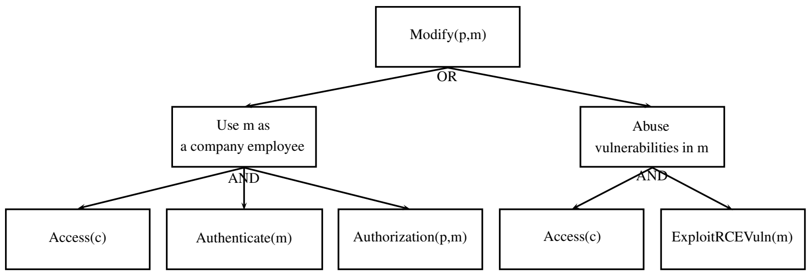

Figure 7: Template Modify(p,m).

Gain access to a component.
The tree built by template *Access(Component)* is shown in Figure 8. The following elements from the system model are taken into account:

- Which network n the component c is part of.
- Which other components x, y, ... are in network n.
- From which components c', ... in other networks n', ... a remote connection to c would be possible.
- Whether the firewall is configured to allow communication from component c' to cross the border between n and n'.

To gain access to a component, the attacker must either gain remote or physical access to it. The physical access leaf will check the attacker's access set. To gain remote access, the attacker must either have physical access to other components in the network of c, which will be checked by the template in Figure 9, or he must access the component from another network. In the latter case, the attacker has to get through the firewall(s) between the two networks, which is checked in the system model.

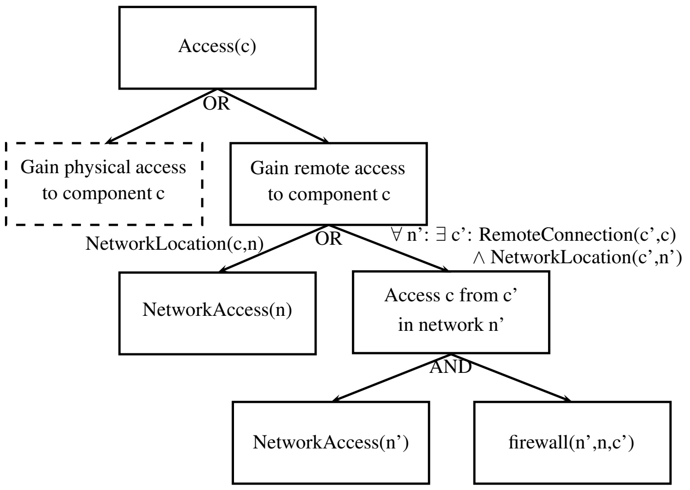
Access(c)
OR
Gain physical access to component c
Gain remote access to component c
NetworkLocation(c,n)
OR
∀ n': ∃ c': RemoteConnection(c',c) ∧ NetworkLocation(c',n')
NetworkAccess(n)
Access c from c' in network n'
AND
NetworkAccess(n')
firewall(n',n,c')

Figure 8: Template Access(c).

Gain access to a network.
The tree built by template *NetworkAccess(Network)* is shown in Figure 9. The following elements from the system model are taken into account:

- Which components c are in network n.

To gain access to a network, the attacker must gain physical access to one of the components located in the network. The physical access leaves will check the attacker's access set.

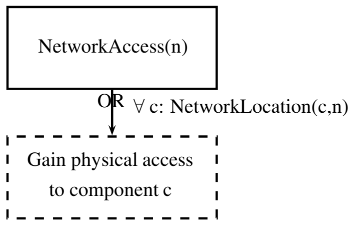
NetworkAccess(n)
OR
∀ c: NetworkLocation(c,n)
Gain physical access to component c

Figure 9: Template NetworkAccess(n).

Authenticate to a software module.
The tree built by template *Authenticate(Module)* is shown in Figure 10. The following elements from the system model are taken into account:

- Which software modules m', m'', $\ldots$ a user must authenticate to before he can access module m.

It is possible that a software module requires authentication to other modules before it can be reached. For instance, in order to access software on an HMI, one must first log on to the operating system module of the machine. This template looks at the system model to infer the chain of modules that leads to module m and generates the tree accordingly. These nodes are recursively matched with the Authenticate template. The attacker must also be able to provide the necessary credentials, which is checked by the LogIn template in Figure 11.

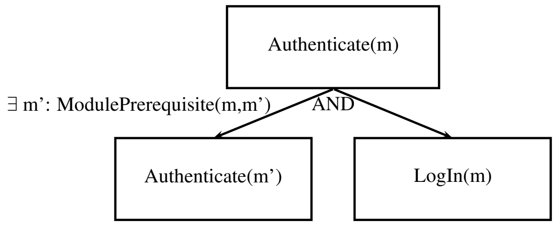

Figure 10: Template Authenticate(m).

Log in to a software module.
The tree built by template *LogIn(Module)* is shown in Figure 11. The following elements from the system model are taken into account:

- Which credentials are required to log in to module m, if there are any.

To log in to a module, the attacker must present the correct credential. The system will look up which credential t is required. It is possible that the attacker already possesses this credential, e.g. when an internal adversary is modelled. Hence a leaf is generated which will check the set of attacker credentials. If not, the attacker must obtain the credential, which is checked by the template in Figure 12. Alternatively, the attacker could attempt to exploit a compromised authentication vulnerability in the module, which is handled by templates similar to the ones in Figures 4 and 5.

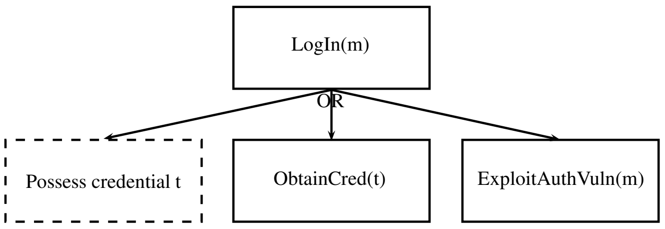

Figure 11: Template Login(m)

Obtain a credential.
The tree built by template *ObtainCred(Credential)* is shown in Figure 12. The following elements from the system model are taken into account:

- Which users contain credential t, if there are any.

To obtain a credential, the system will first check which users possess this credential. The attacker can attempt to steal the credential from one of these users. Alternatively, the attacker could obtain the credential from one of the components where it is stored. This is checked by template 2

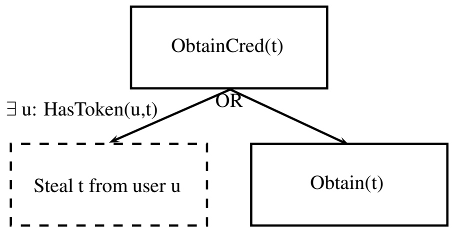

Figure 12: Template ObtainCred(t)

Get authorization to modify a process parameter from a software module.
The tree built by template *Authorization(Parameter, Module)* is shown in Figure 13. The following elements from the system model are taken into account:

- Which credentials have the required permissions associated with them to modify parameters from module m.

To get authorization to modify a parameter from a module, the attacker must present the correct credential. The system will look up which credential t is required. It is possible that the attacker already possesses this credential, e.g. when an internal adversary is modelled. Hence a leaf is generated which will check the set of attacker credentials. If not, the attacker must obtain the credential, which is checked by the template in Figure 12. Alternatively, the attacker could attempt to exploit a privilege escalation vulnerability in the module, which is handled by templates similar to the ones in Figures 4 and 5.

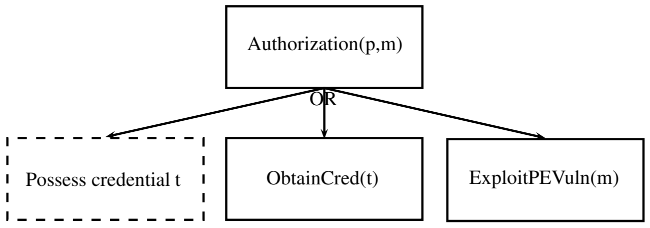

Figure 13: Template Authorization(p,m)