# Conceptual Abstraction of Attack Graphs - a Use Case of securiCAD[*]

Xinyue Mao[1], Mathias Ekstedt[1], Engla Ling[1], Erik Ringdahl[2], and Robert Lagerström[1]

[1] KTH Royal Institute of Technology, Sweden
{xinyuem, mekstedt, englal, robertl}@kth.se
[2] Foreseeti AB, Sweden
erik.ringdahl@foreseeti.com

**Abstract.** Attack graphs quickly become large and challenging to understand and overview. As a means to ease this burden this paper presents an approach to introduce conceptual hierarchies of attack graphs. In this approach several attack steps are aggregated into abstract attack steps that can be given more comprehensive names. With such abstract attack graphs, it is possible to drill down, in several steps, to gain more granularity, and to move back up. The approach has been applied to the attack graphs generated by the cyber threat modeling tool securiCAD.

**Keywords:** Attack Graph · Conceptual Modeling · Cognitive Simplification · securiCAD

## 1 Introduction

The complexity and size of IT systems are growing and as a result so are the attack graphs that represents possible attacks against them. It is important to make sure that the attack graphs are useful and easy to interpret even as they grow. This short paper presents a solution to the problem of visualizing large attack graphs by using abstractions.

This work was driven by a need to simplify the attack graphs generated in the attack simulation tool securiCAD [2]. In securiCAD attack graphs are generated using a fixed attack step library and graph generation logic encoded in a domain specific language. The visualization of the generated graphs follows that same terminology of the semantic level of the library. As this language is quite extensive the generated attack graphs quickly become complex and difficult to grasp, as seen for example in Figure 1. This paper describes a solution to generate abstracted visualizations of attack graphs with two objectives: 1) the abstraction should be formally sound and reversible, and 2) the abstractions should be understandable and make sense to the users of the attack graph. The

[*] This work has received funding from the Swedish Civil Contingencies Agency through the research centre Resilient Information and Control Systems (RICS) as well as the SweGRIDS competence center.

the paper contributes both with an approach to form visually simplified attack graphs particularly from asset-based attack graph formalisms, as well as a case study on the securiCAD tool where the approach was applied. The approach is limited to simplifying already generated attack graphs. Consequently, the question of the correctness of these attack graphs or the computational challenges of computing them is outside the scope of this paper.

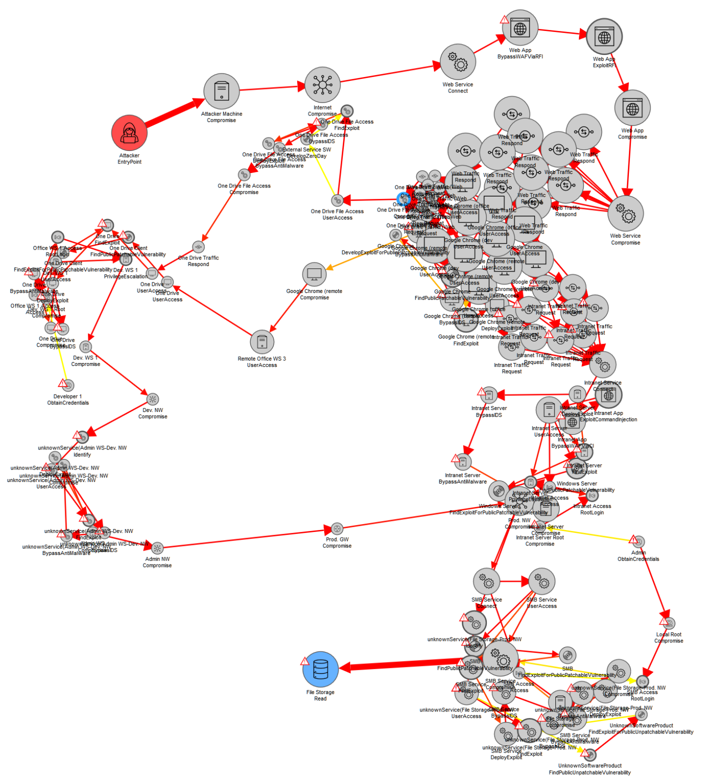

**Fig. 1.** An example of a visually complex attack graph in securiCAD

## 2 Related Work

The problem of making information more accessible in complicated and large attack graphs can be solved by several different methods as outlined in the attack graph taxonomy by Kaynar [8]. One method of simplifying is, for example, hierarchical division. In our solution the hierarchical division is derived from the securiCAD syntax. In the tool CyGraph [14], which uses multiple data sources to construct an attack graph, the different data sources are used as layers. The layers are, for instance, the network infrastructure layer and the cyber threats layer. Other research has based the division on attribute values or the connectedness of the attack graph [15]. Another example of division is HARM, a hierarchical attack representation model [4]. HARM divides the graph in two hierarchical levels, the network and the vulnerability levels.

An alternative solution to simplify attack graphs is to aggregate parts of the graph. The aggregation can be achieved by grouping the attack steps or network objects [9]. However, this work was presented as a suggested approach of reducing the complexity but did not present a method. The approach was also intended to reduce the complexity of generating graphs, rather than presenting them. Please note that, in this paper, we do not aim to reduce the complexity of graph generation. This paper is concerned with simplifying attack graphs' representation. Homer et al. presented another approach to aggregation where they remove attack steps that they consider "useless" [3]. Their definition of "useless" is that the attack step is not necessary to understand the security vulnerability.

The Network Security Planning Architecture (NetSPA) uses a method of pruning to simplify the representation of an attack graph [1]. The user can prune the attack graph by choosing a specific goal state and only visualize the attack steps that ends with that specific state. This method of pruning an attack graph has not been implemented in the solution presented in this paper.

A common denominator for some of the identified related work is that the aggregation is performed according to fixed rules built upon the specific attack graph elements. In our work we present a solution for building aggregation patterns (that can be changed over time) as well as a specific set of patterns for the securiCAD tool.

The patterns that are constructed in our work are named according to a visualization vocabulary. The intention of these patterns is to align with established terminology. Examples of such resources are MITRE's Common Attack Pattern Enumeration and Classification (CAPEC) [13] and ATT&CK matrix [12]. However, in our work a bespoke vocabulary was developed matching the securiCAD tool.

There are methods of reducing the complexity of graphs' visual representations in ways not used in the solution presented in this paper. By working with for example different thickness or colors of lines, it is possible to add more information to the graph without adding more items [10].

Finally, there are methods for reducing attack graph complexity for other reasons than visualization. These reasons can be, for instance, to reduce computational complexity as seen in the survey by Hong et al. [5]. However, these related works are not included because they fall out of scope of this paper.

## 3 securiLang

The securiCAD tool [2] is generating its attack graphs according to the logic of a domain specific language called securiLang. In brief the securiLang consists of `Assets` (e.g. `Network`, `Host`, `Service`, `Data flow`) that can have `Associations` to each other (e.g. a `Host` can either `Root execute` or `User execute` a `Service`). To `Assets` there are `Attack steps` associated (e.g. a `Data flow` can be `Eavesdropped`, `Replayed`, or `DoSed`). In addition, `Assets` also have `Defenses` associated. However, this paper is only looking at attack graphs, not the full defense graphs, so they are not further discussed here. The full list of assets and their associated attack steps are presented in Appendix A.

Furthermore, securiLang contains rules for how attack graphs are generated when instance models are built following the language. Attack steps have potential parent and child steps, depending on how the instance models is constructed. E.g. Figure 2 demonstrates the relation between asset `Access Control` and its attack step `AccessControl.Access`. A potential parent to this step is `Host.UserAccess`, however this is only true if the `Access Control` asset is having an `Authorization` association to the `Host` asset. If the `Access Control` `Authorizes` a `Service` instead it will be the `Service.Connect` that leads to `AccessControl.Access`, and so on. In this example `AccessControl.Access` only has child attack steps located at the `Access Control` asset.

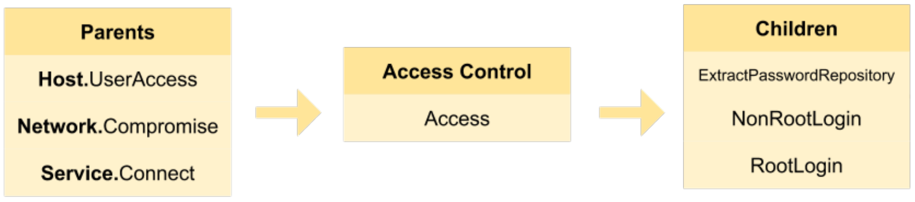

**Fig. 2.** An example of Access Control showing the association of an attack step

The full logic in terms of association traversal and OR/AND attack step dependencies are omitted here to avoid complexity unnecessary for the purpose of the paper. Furthermore, the securiCAD tool is conducting probabilistic calculations of the attack graphs and aggregates a time-to-compromise value over all possible attack paths enabled by an instance model which quickly leads to a myriad of attack vectors. However, these dimensions of the attack graph generation and calculation is not discussed here since the attack graph aggregation suggested in this paper follows the same structure for all branches of an attack graph. In the next chapter we will go into the abstraction mechanism.

## 4 Attack Graph Abstraction

Our abstraction mechanism addresses attack graph languages based on relational models following our previous work such as [6], [16] and [7]. The aggregation approach is illustrated in the conceptual model in Figure 3.

The original attack graph language is depicted in the yellow classes. We consider this the level 0. From the meta model of this language we assume three things; that it includes an `Attack Step` class, an `Asset` class, and an `Association` relating the two classes. This is true for our previous work, but the intention is to leave as few requirements on the original language as possible[3]. Not strictly needed to apply the approach, but later used in our use case for convenience, the Figure also illustrates that `Assets` can have `relationships` between each other. In the original language these relationships are however instantiated so that e.g. a *Host* `Asset` *executes* a *Client* `Asset`.

On top of this original language we introduce two new classes; the `Abstract Attack Step` and the `Abstract Asset`. The purpose of the former class is to enable aggregation of `Attack Steps` and `Abstract Attack Steps` in leveled hierarchies. Thus we also introduce an `Aggregation` relationship between `Abstract Attack Steps` and `Attack Steps` as well as a self reference for `Abstract Attack Steps`. Here we follow the standard semantics of aggregation, for instance used in UML, where the aggregator is nothing more than, and cannot exist without, its constituents. More precisely, we introduce two types of aggregations (not illustrated separately in the Figure to avoid clutter); `Mandatory Aggregation` and `Optional Aggregation` since not always are all possible sub steps present in the illustrated attack graph[4].

In order to create the aggregation hierarchies, we introduce a `Level` variable. We stipulate that the original language is level 0 and then for every introduced aggregation the `Level` is increased. However, we do not restrict aggregation so that all constituents need to be of the same `Level` and consequently the `Level` value has to be determined by the highest constituent `Level`. The aggregator `Level` must be increased with at least 1 but we also allow to add any (natural) number in order for the aggregation designer to build "conceptually even" layers. Similarly to the original language we also stipulate that `Abstract Attack Steps` must have an `Association` to an `Asset` or an `Abstract Asset`. With the `Abstract Asset` we want to enable the possibility to associate `Abstract Attack Steps` with any type of asset that is deemed useful. And just as we assume that `Attack Steps` and `Assets` come with sort of `Names` (or identifier) from the original language, also the `Abstract Attack Steps` and `Abstract Assets` are given `Names`.

Given these rules for abstracting attack graphs there are many patterns imaginable for how this can be done. In the case study presented in this paper mainly

[3] Obviously the exact naming of the classes and their relationship is not relevant.

[4] This could be due to the attack steps being aggregated by an OR attack step, but it could also be that not all attack vectors are displayed at the same time. In securiCAD for instance only highly probable attack vectors are illustrated.

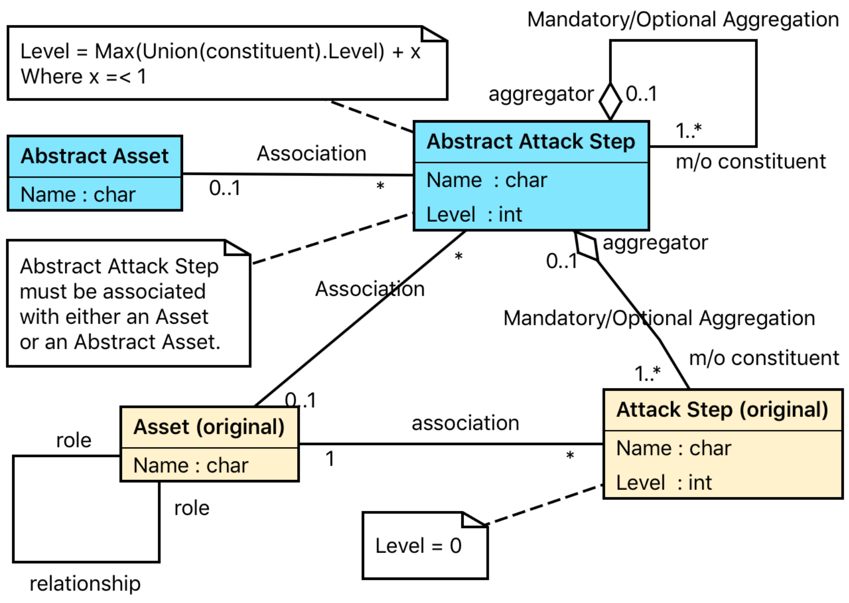

**Fig. 3.** Attack graph abstraction overview

two patterns have been used. Firstly, aggregation of a fixed series of parent/child attack steps following on each other according to the logic of the original language. An example of such a case is the one illustrated in in Figure 2. Secondly, we have based aggregations on the `Asset` they are `Associated` with. An `Abstract Attack Step` is defined that represents a full compromise of the `Asset`. Here we want to aggregate all `(Abstract) Attack Steps` that relate to this particular `Asset`. This pattern allows for conceptually organizing the `Assets` into the layers, so that one `Asset`'s full compromise `Abstract Attack Step` is then aggregated into another `Asset`'s full compromise `Abstract Attack Step`.

## 5 Abstracting securiLang

Above we have presented the general mechanisms devised for abstracting model based attack graphs. We will now move on to describing a suggested abstraction for securiLang [5]. In this work we have devised six abstraction levels, three attack step based abstractions(Level 1-3) and three asset based abstractions(Level 4-6).

For each level of abstraction, we merge the original attack steps into one abstracted step according to the patterns introduced in the last section. All securiLang abstractions made are found in Appendix B. To illustrate the work

[5] The work is based on the securiLang version contained in securiCAD v1.4.

we take `Exploit vulnerabilities and compromise Client` as an example to explain how the abstracting operation works. Table 1 contains a snippet from the full table in Appendix B.

| Original Attack Steps | 1st Level | 2nd Level | 3rd Level |
|---|---|---|---|
| Client. FindExploit | O | Find exploit | Exploit vulnerabilities and compromise Client |
| Client. DeployExploit | O | Deploy exploit | |
| Client. BypassIDS | | | |
| Client. BypassAntiMalware | | | |
| Client. Compromise | O | Compromise Client | |

**Table 1.** Abstraction of attack steps in asset Client

The 'O' in Table 1 means that in this level, all the abstracted steps remain the same with former level. For example, the 'O' in the first line of Figure 1 means the same with the name in former level, in this case, this 'O' equals to `Client.FindExploit`. At the second level, the single steps are translated to `Find exploit` and `Compromise client`, while the three steps in the middle are grouped into Deploy exploit. Then in the third level, the abstracted steps in the second level can be merged into `Exploit vulnerabilities and compromise Client`. Figure 4 illustrates the abstraction in Table 1 graphically.

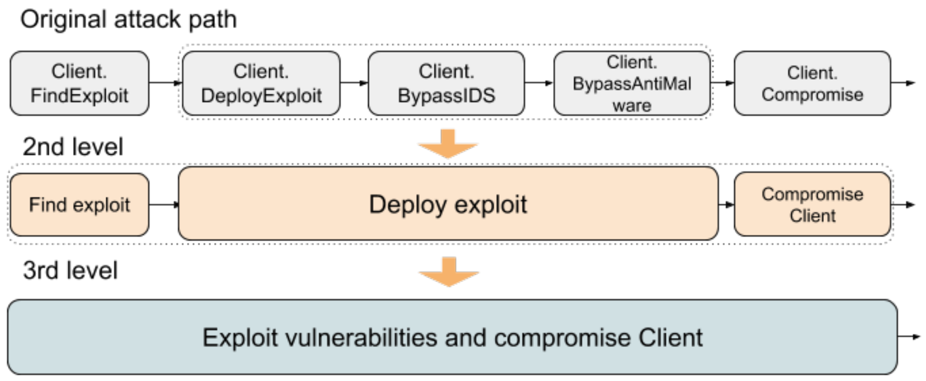

**Fig. 4.** Abstracted attack step of Access Control from original attack path to Level 3

Semantically `Find exploit` denotes that attacker is able to find an exploit to use on a `Client`. `Deploy exploit` means that the deployment of found and

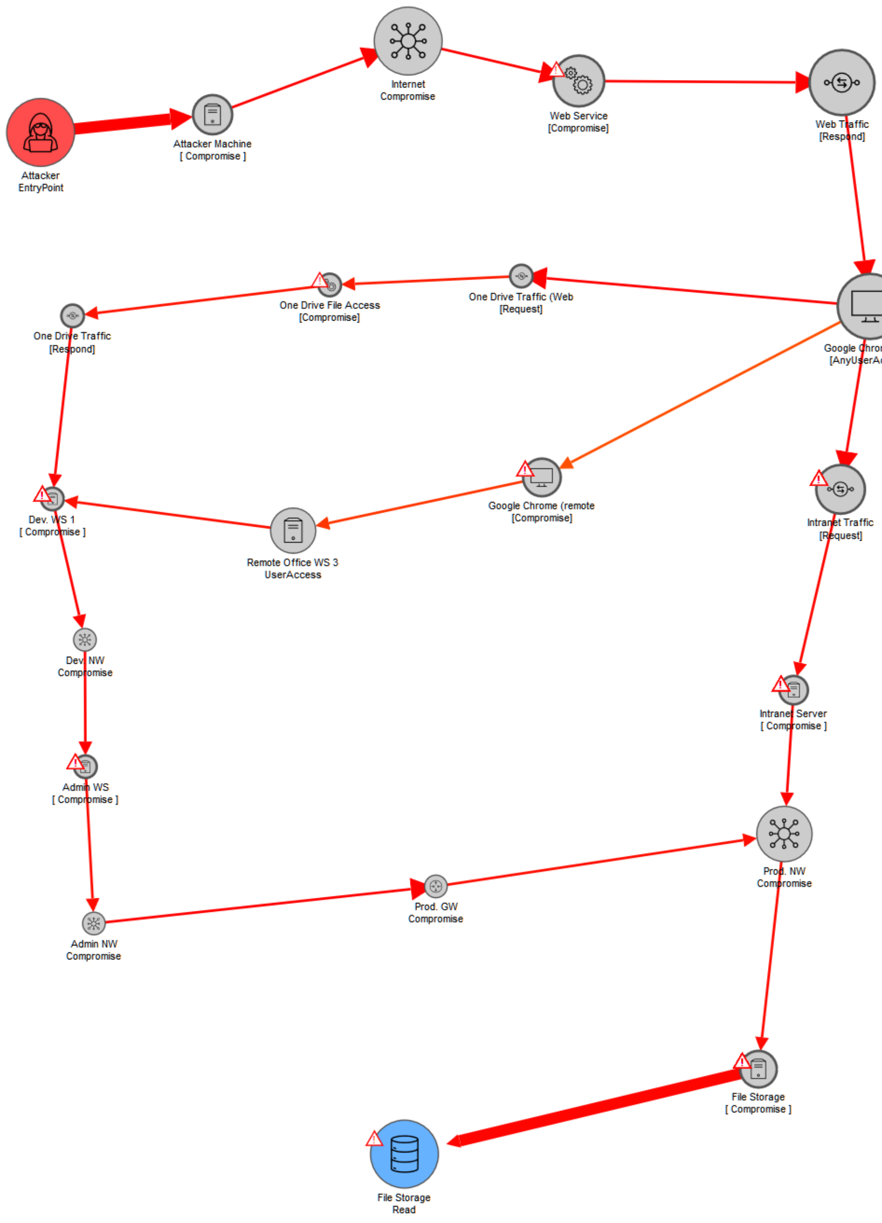

**Fig. 5.** An example of an abstracted attack graph in securiCAD on Level 4

developed exploits for the `Client`'s `Software Product`. After a successfully executed exploit is deployed, bypassing some protection mechanisms, the `Client` is `Compromised`. When making the higher level abstractions we want the terminology to be as intuitive and self explaining as possible, in this case exploit can be explained as taking advantages of a bug or a vulnerability, which is not captured in the original securiLang terminology. Thus, in this case level 3 abstract attack step name was chosen as `Exploit vulnerabilities and compromise Client`. On level 2 we find examples of the use of the same name as used in the Level 0 language, and in two instances the semantics is kept, while in one case the semantics is changed (expanded so that also defense bypassing is included in the `Deploy exploit` abstract attack step).

Altogether we have done four asset based abstractions; `Client Compromise` and `Service Compromise` are found on Level 4, `Host Compromise` on Level 5, and `Network Compromise` is on the 6th and final Level. In securiLang these four assets; `Client`, `Service`, `Host`, and `Network`, each attack steps called `Compromise`, thus it was straight forward to develop new new `Abstract Attack Step` with the same name as the original language new `Attack Step` to create these levels.

As an illustration of the end result of this work it has enabled us to simplify the visually complex attack graph in Figure 1 to an attack graph as seen in Figure 5, on Level 4.

## 6 Evaluating the securiLang Abstraction

In order to evaluate our design of the securiLang abstraction we were addressing the following quality criteria (relating to our second objective in the Introduction):

1. The vocabulary chosen as representation provides associations that accurately reflects the attack steps on level 0.
2. Each of the representations are made so that the levels feels conceptually even in terms of level of perceived detail.
3. The design of the six levels are perceived as intuitive.

The evaluation was made in two phases; firstly two senior security experts that have previously worked with securiCAD (but not involved in the abstraction project) were interviewed and secondly a survey was sent out to cyber security students that had not encountered the tool.

### 6.1 The interviews

For the interviews, we used a questionnaire where the respondents were asked to rate statements and answer questions. To address the first evaluation criterion statements such as "*Root login to Host* well describes the original attack process?" were posed with answering options on a five-level scale from *Strongly disagree* to *Strongly agree*. For the second criterion, questions such as "Do you agree that *Access* and *Root login* can be merged into one attack step in the 1st level?" were posed with the same answering option. Both statements were followed by an open question to allow the interviewee to comment freely on their answer.

In general, the two respondents gave the original design a good evaluation. However, several smaller changes were suggested and the model presented in this paper is the refined version after this input. For instance, `Use legitimate access` used to be called `Extract password from user`, which represented the two attack steps `ExtractFromUser` and *Compromise*. Another example is *FindExploit* and `DeployExploit` that were originally grouped together, while `BypassIDS`, `BypassAntiMalware` and `Compromise` were aggregated. The respondents instead suggested that `DeployExploit`, `BypassIDS` and `BypassAntiMalware` are grouped into one abstracted step instead.

## 6.2 The survey

An online survey was created where participants rated the attack steps representations from strongly disagree to strongly agree. Altogether 24 students with major in information technology and at least basic skills in cyber security participated.

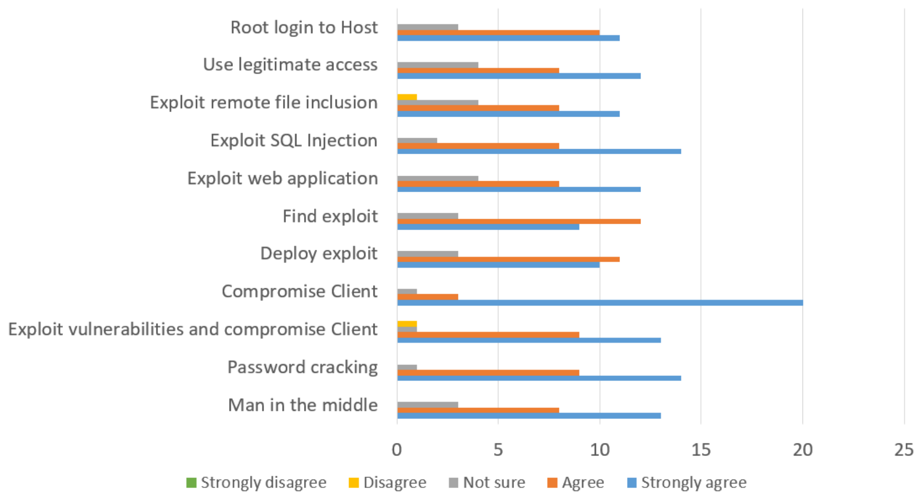

**Fig. 6.** Results of survey of single abstracted representation with corresponding numbers

Related to our first and third criteria, here the participants rated eleven selected attack step abstractions based on their intuitive perception and understanding, ranging from strongly disagree to strongly agree. Figure 6 shows the result of the survey for the different attack step abstractions. All representations except for `Find exploit` and `Deploy exploit` were rated with *Strongly agree* as the most popular choice. Overall only few *disagreed* and no participant chose *Strongly disagree*. When observing the results for *Agree* and *Not sure*, the number of people in these categories are evenly distributed among all the representations. Thus, it is concluded that each representation gets almost the consistent recognized level. For all representations, approximately 90% of the participants have chosen agree or strongly agree. This shows that most people agree with attack steps representations, believing they can well describe the original attack process and provide an easily understandable interactive view, which could meet the first and third criterion.

## 7 Future Work

This project reports on early and exploratory work. Consequently it is natural that several avenues for future work has been identified from this project. From a practical point of view making the abstraction a fully maintainable and configurable capability is the most important for further adoption of the idea. First of all, securiLang (and any other asset based attack graph language) evolves over time with new types of attacks and assets, so likely the attack aggregations also need to be updated. Moreover, securiCAD supports several different languages and each would need to have its language specific attack aggregations.

We can also envision that one would like to abstract the same attack graph according to different patterns depending on who is looking at it and for what purposes. In the case study presented here the aggregation was driven mainly by the experiences at the tool vendor of some attack path segments had been found difficult to explain to tool users in combination with some gut feeling of nice-to-have features. The aggregation could also be addressed e.g. from an alignment point of view where the aggregations are devised to map some more well established terminology such as Mitre's CAPEC [13] and ATT&CK [12], or Lockheed Martin's Kill Chain concept. Thus ongoing work is now to develop a configuration structure that enables to define the conceptual attack graph abstractions for any language in a format that is both readable by securiCAD and easy to expand and understand as a language designer. As of now, the configuration structure is in essence a single JSON file that for each level of abstraction defines a set of rules which are applied sequentially on the attack graph, such as the below code snippet.

```
Level 2: {
 Client: [
   {
     collect: [
       Client.DeployExploit,
       Client.BypassIDS,
       Client.BypassAntiMalware
     ],
     replacement: {
       class: Client,
       attackstep: Deploy exploit
     }
```

```
    }
  ]
}
```

From a more theoretical point of view, formalizing the aggregation approach suggested here into a (meta) language with formal semantics is a natural next step. Devising and structuring design patterns for aggregation can further be of interest. In addition, adding configuration features for syntax and form editing (coloring, sizing, etc.) is likely to improve attack graph readability significantly.

## 8 Conclusion

The purpose of this work was to address the problem of attack graphs that are difficult to understand as they grow large and complex. Specifically, the attack graphs generated by the tool securiCAD constituted a case study. An overall approach was suggested and applied in the case study. The results are generally positive. Firstly the attack graph abstraction approach was found viable and useful for conducting the case study. Secondly, the abstraction patterns developed in the case study were also promising. Even though the evaluation of the case study patterns was rather weak from a methodological standpoint at least all indications suggest that the patterns are fit for its intended purposes. Also the intuition at the case study company is that this is important correct enough to further develop the idea and implement support for it in the securiCAD tool. But to come to a more universal conclusion on the appropriateness of these particular aggregation patterns would obviously require more validation studies.

And for the generic conceptual abstraction approach per se has only been tested with a single case study. Obviously this is far from a full validation of the approach. Analytically we can however conclude that the approach is primarily suitable for use cases where the attack graphs we want to abstract follow predictable patterns since aggregation patterns need to be created manually.

Furthermore, the case study does not make use of `Abstract Assets` class. To use introduce for instance highly abstract assets as means for abstraction seems however reasonable. E.g. for large ICT infrastructures it could be interesting to. introduce an `Abstract Asset` called `Zone` onto which one or several `Network.compromise` could be aggregated into an `Abstract Attack Step` labelled `Zone.compromise`.

This paper is based on and extends the work presented in the Master thesis by Mao [11].

## References

1. Michael Lyle Artz. *NetSPA: A Network Security Planning Architecture.* Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2019.

2. Mathias Ekstedt, Pontus Johnson, Robert Lagerström, Dan Gorton, Joakim Nydrén, and Khurram Shahzad. Securicad by foreseeti: A cad tool for enterprise cyber security management. In *2015 IEEE 19th International Enterprise Distributed Object Computing Workshop*, pages 152–155. IEEE, 2015.
3. John Homer, Ashok Varikuti, Xinming Ou, and Miles A. McQueen. Improving attack graph visualization through data reduction and attack grouping. In *Visualization for Computer Security*, pages 68–79. Springer Berlin Heidelberg, 2008.
4. Jin Hong and Dan Kim. Harms: Hierarchical attack representation models for network security analysis. *Australian Information Security Management Conference*, 12 2012.
5. Jin B. Hong, Dong Seong Kim, Chun Jen Chung, and Dijiang Huang. A survey on the usability and practical applications of graphical security models. *Comput. Sci. Rev.*, 26(C):1–16, November 2017.
6. Pontus Johnson, Robert Lagerström, and Mathias Ekstedt. A meta language for threat modeling and attack simulations. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, page 38. ACM, 2018.
7. Pontus Johnson, Alexandre Vernotte, Mathias Ekstedt, and Robert Lagerstrm. pwnpr3d: An attack-graph-driven probabilistic threat-modeling approach. In *2016 11th International Conference on Availability, Reliability and Security (ARES)*, pages 278–283, 2016.
8. Kerem Kaynar. A taxonomy for attack graph generation and usage in network security. *Journal of Information Security and Applications*, 29:27 – 56, 2016.
9. Igor Kotenko and Mikhail Stepashkin. Attack graph based evaluation of network security. In *Communications and Multimedia Security*, pages 216–227. Springer Berlin Heidelberg, 2006.
10. Eric Li, Jeroen Barendse, Frederic Brodbeck, and Axel Tanner. From a to z: Developing a visual vocabulary for information security threat visualisation. In *Graphical Models for Security*, pages 102–118. Springer International Publishing, 2016.
11. Xinyue Mao. Visualization and natural language representation of simulated cyber attacks. Master's thesis, KTH Royal Institute of Technology, 2018.
12. MITRE. About ATT&CK. `https://attack.mitre.org/`, 2018. Accessed 2019-04-01.
13. MITRE. About CAPEC. `https://capec.mitre.org/about/index.html`, 2018. Accessed 2019-03-25.
14. Steven Noel, Eric Harley, Kam Him Tam, Michael Limiero, and Matthew Share. Chapter 4 - cygraph: Graph-based analytics and visualization for cybersecurity. In *Cognitive Computing: Theory and Applications*, volume 35 of *Handbook of Statistics*, pages 117 – 167. Elsevier, 2016.
15. Steven Noel and Sushil Jajodia. Managing attack graph complexity through visual hierarchical aggregation. In *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*, pages 109–118. ACM, 2004.
16. Teodor Sommestad, Mathias Ekstedt, and Hannes Holm. The cyber security modeling language: A tool for assessing the vulnerability of enterprise system architectures. *IEEE Systems Journal*, 7(3):363–373, 2013.

## A Appendix securiLang assets and attack steps

| Assets | Attack steps |
|---|---|
| AccessControl | Access, ExtractPasswordRepository, NonRootLogin, RootLogin |
| Client | BypassAntiMalware, BypassIDS, Compromise, DenialOfService, DeployExploit, FindExploit, UserAccess |
| Dataflow | Access, DenialOfService, Eavesdrop, ManInTheMiddle, Replay, Request, Respond |
| Datastore | Delete, Read, Write |
| Firewall | Compromise, DiscoverEntrance |
| Host | ARPCachePoisoning, BypassAntiMalware, BypassIDS, Compromise, DenialOfService, DeployExploit, FindExploit, PhysicalAccess, PrivilegeEscalation, USBAccess, UserAccess |
| Keystore | Delete, Read |
| Network | ARPCachePoisoning, Compromise, DNSSpoof, DenialOfService |
| Router | Compromise, DenialOfService, Forwarding |
| Service | ApplicationLogin, BypassAntiMalware, BypassIDS, Compromise, Connect, DenialOfService, DeployExploit, FindExploit, NonRootShellLogin, RootShellLogin, UserAccess |
| SofwareProduct | DevelopExploitForPublicPatchableVulnerability, DevelopExploitForPublicUnpatchableVulnerability, DevelopZeroDay, FindExploitForPublicPatchableVulnerability, FindExploitForPublicUnpatchableVulnerability, FindPublicPatchableVulnerability, FindPublicUnpatchableVulnerability |
| UserAccount | Compromise, ExtractFromUser, GuessOffline, GuessOnline |
| WebApplication | BypassWAFViaCI, BypassWAFViaRFI, BypassWAFViaSQLInjection, BypassWAFViaXSS, DiscoverNewVulnerability, ExploitCommandInjection, ExploitRFI, ExploitSQLInjection, ExploitXS |

securiLang is further described at https://community.securicad.com/securilang-reference-manual/

## B Appendix Abstraction patterns of securiLang

| Asset | Attack Step | Attack Steps Based Abstraction | | | Assets Based Abstraction | | |
|---|---|---|---|---|---|---|---|
| | | 1st Level | 2nd Level | 3rd Level | 4th Level | 5th Level | 6th Level |
| **Access Control** | Access | Root login to Service / Host | X | X | X | X | X |
| | **UserAccount**(root).Compromise | | | | | | |
| | RootLogin | | | | | | |
| | Access | Non-root login to Service / Host | X | X | X | X | X |
| | **UserAccount**(non-root).Compromise | | | | | | |
| | NonRootLogin | | | | | | |
| | Access | Password cracking | X | X | X | X | X |
| | ExtractPasswordRepository | | | | | | |
| | **UserAccount**.GuessOffline | | | | | | |
| | **UserAccount**.Compromise | | | | | | |
| **User Account** | ExtractFromUser | Use legitimate access | X | X | X | X | X |
| | Compromise | | | | | | |
| | **AccessControl**.Access | Guess poissible credentials | X | X | X | X | X |
| | GuessOnline | | | | | | |
| | Compromise | | | | | | |
| **Client** | UserAccess | Access to Client | X | Exploit vulnerbilities and compromise Client | Client Compromise | | |
| | **SoftwareProduct**.FindExploit / DevelopExploit / DevelopZeroDay / FindEndOfLifeVuln | | | | | | |
| | FindExploit | O | Find exploit | | | | |
| | DeployExploit | O | Deploy exploit | | | | |
| | BypassIDS | | | | | | |
| | BypassAntiMalware | | | | | | |
| | Compromise | O | Compromise Client | | | | |
| | **Host**.DenialOfService | Denial of Service | X | X | | | |
| | DenialOfService | | | | | | |
| | **Dataflow**.DenialOfService | | | | | | |
| **Dataflow** | Access | Denial of Service | X | X | X | X | X |
| | **Service**.DenialOfService / **Client**.DenialOfService / **Network**.DenialOfService | | | | | | |
| | DenialOfService | | | | | | |
| | Access | Eavesdrop | X | X | X | X | X |
| | Eavesdrop | | | | | | |
| | Access | Man in the middle | X | X | X | X | X |
| | ManInTheMiddle | | | | | | |
| | Request / Respond | | | | | | |
| | Access | Replay | X | X | X | X | X |
| | Replay | | | | | | |
| | Request | Service.Connect | X | X | X | X | X |
| | Service.Connect | | | | | | |
| | Respond | Client.UserAccess | X | X | X | X | X |
| | Client.UserAccess | | | | | | |
| **Network** | ARPCachePoisoning | ARP Cache Poisoning | X | X | X | X | Network Compromise |
| | DNSSpoof | DNS Spoof | X | X | X | X | |
| | **Router**.DenialOfService | Denial of Service | X | X | X | X | |
| | DenialOfService | | | | | | |
| | **Dataflow**.DenialOfService | | | | | | |

* Steps in grey are optional in the aggregation

| Asset | Attack Step | Attack Steps Based Abstraction | | | Assets Based Abstraction | | |
|---|---|---|---|---|---|---|---|
| | | 1st Level | 2nd Level | 3rd Level | 4th Level | 5th Level | 6th Level |
| **Keystore** | Delete | Damage data | X | X | X | X | X |
| | Datastore.Delete | | | | | | |
| | Read | Compromise UserAccount | X | X | X | X | X |
| | UserAccount.Compromise | | | | | | |
| | Read | Damage data | X | X | X | X | X |
| | Datastore.Read / Datastore. Write | | | | | | |
| **Host** | ARPCachePoisoning | ARP Cache Poisoning | X | X | X | Host Compromise | |
| | UserAccess / AccessControl.Bypass | O | Privilege Escalation | X | X | | |
| | PrivilegeEscalation | | | | | | |
| | Compromise | | | | | | |
| | PhysicalZone.Compromise | PhysicalZone Compromise | X | X | X | | |
| | USBAccess / PhysicalAccess | | | | | | |
| | UserAccess | Access to Host | X | Exploit vulnerabilities and compromise Host | X | | |
| | **SoftwareProduct**.FindExploit / DevelopExploit / DevelopZeroDay / FindEndOfLifeVuln | | | | | | |
| | FindExploit | O | Find exploit | | | | |
| | DeployExploit | O | Deploy exploit | | | | |
| | BypassIDS | | | | | | |
| | BypassAntiMalware | | | | | | |
| | Compromise | O | Compromise Host | | | | |
| **Service** | Connect | Non-root shell login to Service | X | X | Service Compromise | | |
| | **AccessControl**.NonRootLogin | | | | | | |
| | NonRootShellLogin | | | | | | |
| | **Host**.UserAccess | | | | | | |
| | UserAccess | Access to Service | X | Exploit vulnerbilities and compromise Service | | | |
| | **SoftwareProduct**.FindExploit / DevelopExploit / DevelopZeroDay / FindEndOfLifeVuln | | | | | | |
| | FindExploit | O | Find exploit | | | | |
| | DeployExploit | O | Deploy exploit | | | | |
| | BypassIDS | | | | | | |
| | BypassAntiMalware | | | | | | |
| | Compromise | O | Compromise Service | | | | |
| | Connect | Root shell login to Service / Host | X | X | | | |
| | **AccessControl**.RootLogin | | | | | | |
| | RootShellLogin | | | | | | |
| | Service(non-root).Host.UserAccess / Host | | | | | | |
| **Router** | **Firewall**.DiscoverEntrance | Router Forwarding | X | X | X | X | X |
| | Forwarding | | | | | | |
| **Datastore** | Write | Damage data | X | X | X | X | X |
| | Delete | | | | | | |

* Steps in grey are optional in the aggregation

| Asset | Attack Step | Attack Steps Based Abstraction | | | Assets Based Abstraction | | |
|---|---|---|---|---|---|---|---|
| | | 1st Level | 2nd Level | 3rd Level | 4th Level | 5th Level | 6th Level |
| **SoftwareProduct** | FindPublicPatchableVulnerability / FindPublicUnpatchableVulnerability | Deploy Exploit for Public Patchable Vulnerbility | X | X | X | X | X |
| | DevelopExploitForPublic PatchableVulnerability | | | | | | |
| | FindPublicUnpatchableVulnerability / FindPublicPatchableVulnerability | Find Exploit for patchable / unpatchable vulnerbility | X | X | X | X | X |
| | FindExploitPublicUnpatchableVulnerabil ity /FindExpllitPublicPatchableVulnerbility | | | | | | |
| | **Host./Service./Client.**UserAccess | Develop Zero Day | X | X | X | X | X |
| | DevelopZeroDay | | | | | | |
| | FindPublicUnpatchableVulnerability | Deploy Exploit for Unpatchable Vulnerbility | X | X | X | X | X |
| | DevelopExploitForPublicUnpatchableVu lnerability | | | | | | |
| **Web Application** | DiscoverNewVulnerability | Exploit Command Injection | Exploit web application | X | X | X | X |
| | Service.connect | | | | | | |
| | BypassWAFViaCI | | | | | | |
| | ExploitCI | | | | | | |
| | DiscoverNewVulnerability | Exploit Remote File Inclusion | | | | | |
| | Service.connect | | | | | | |
| | BypassWAFViaRFI | | | | | | |
| | ExploitRFI | | | | | | |
| | DiscoverNewVulnerability | Exploit SQL Injection | | | | | |
| | Service.connect | | | | | | |
| | BypassWAFViaSQLInjection | | | | | | |
| | ExploitSQLi | | | | | | |
| | DiscoverNewVulnerability | Exploit Cross Site Scripting | | | | | |
| | Service.connect | | | | | | |
| | BypassWAFViaXSS | | | | | | |
| | ExploitXSS | | | | | | |

* Steps in grey are optional in the aggregation