D^3: Deception, Deterrence, and Disclosure in Cybersecurity

V.S. Subrahmanian

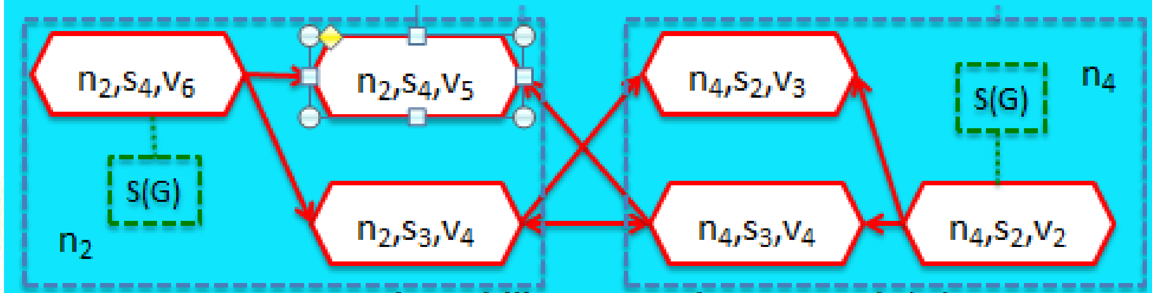
vs@cs.umd.edu

@vssubrah

Joint work with Sushil Jajodia, Noseong Park, and Edoardo Serra

Our D3 Framework

Enterprise network



System Vulnerability Dependency Graph (G)

State – his knowledge of the system

ATTACKER

Can learn !

Deception

Disclosure

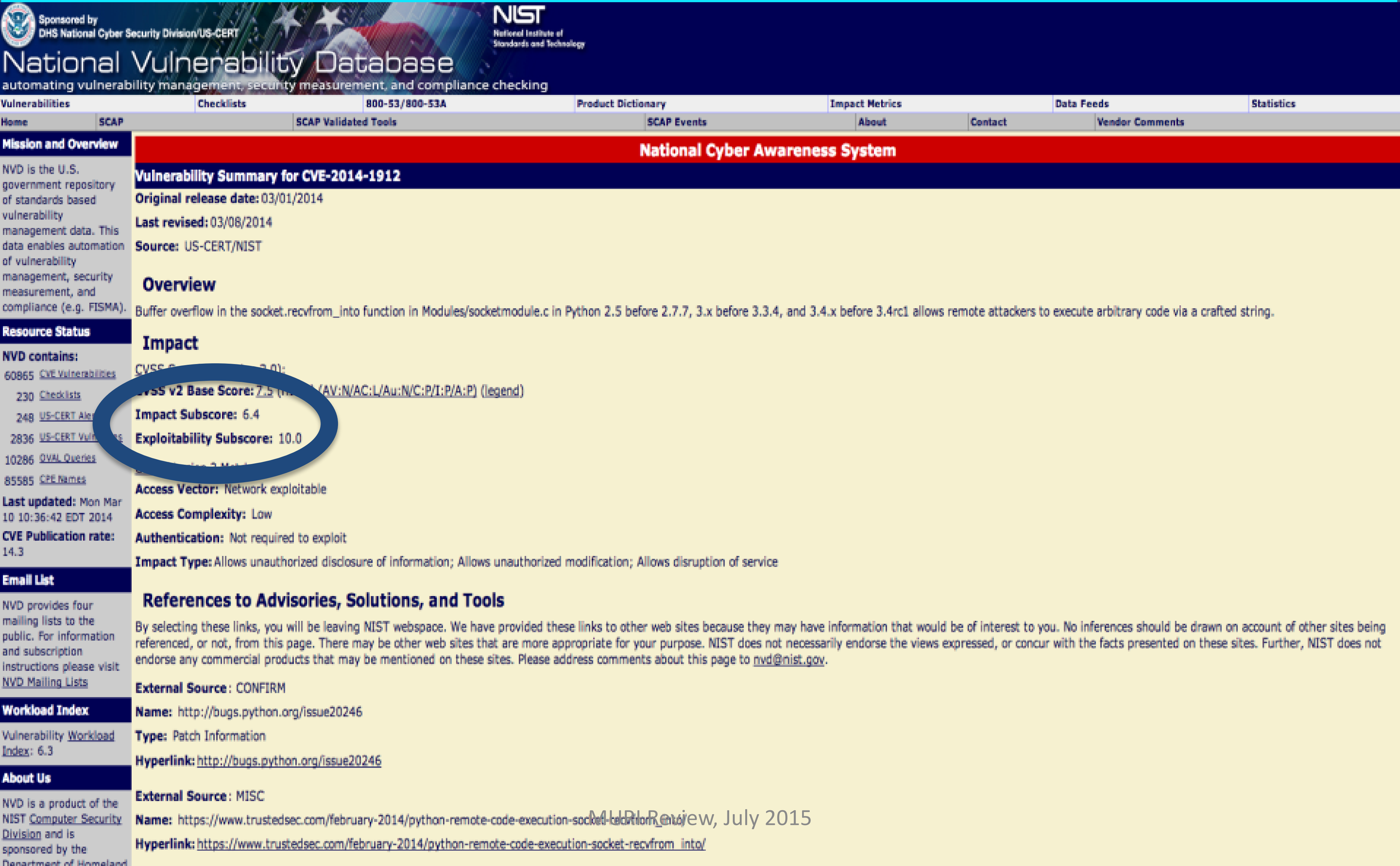
Deterrence

Software and Vulnerabilities

- ***S*** set of software
- ***V*** set of vulnerabilities
- mapping $\boldsymbol{v:S \rightarrow 2^V}$
- for each vulnerability ***v*** in ***V***
  - ***impact(v).*** *Impact on enterprise if the vulnerability is exploited.*
  - ***diff(v).*** *Difficulty of exploiting v.*

Such measures are available through multiple sources, e.g. NIST’s NVD and CVSS, and MITRE’s CWSS

NIST National Vulnerability Database



Sponsored by DHS National Cyber Security Division/US-CERT

NIST National Institute of Standards and Technology

National Vulnerability Database

automating vulnerability management, security measurement, and compliance checking

Mission and Overview

NVD is the U.S. government repository of standards based vulnerability management data. This data enables automation of vulnerability management, security measurement, and compliance (e.g. FISMA).

Resource Status

NVD contains:

60865	CVE Vulnerabilities
230	Checklists
248	US-CERT Ale
2836	US-CERT Vul s
10286	OVAL Queries
85585	CPE Names

Last updated: Mon Mar 10 10:36:42 EDT 2014

CVE Publication rate: 14.3

Email List

NVD provides four mailing lists to the public. For information and subscription instructions please visit NVD Mailing Lists

Workload Index

Vulnerability Workload Index: 6.3

About Us

NVD is a product of the NIST Computer Security Division and is sponsored by the Department of Homeland

National Cyber Awareness System

Vulnerability Summary for CVE-2014-1912

Original release date: 03/01/2014

Last revised: 03/08/2014

Source: US-CERT/NIST

Overview

Buffer overflow in the socket.recvfrom_into function in Modules/socketmodule.c in Python 2.5 before 2.7.7, 3.x before 3.3.4, and 3.4.x before 3.4rc1 allows remote attackers to execute arbitrary code via a crafted string.

Impact

CVSS Severity (version 2.0):

CVSS v2 Base Score: 7.5 (HIGH) (AV:N/AC:L/Au:N/C:P/I:P/A:P) (legend)

Impact Subscore: 6.4

Exploitability Subscore: 10.0

CVSS Version 2 Metrics:

Access Vector: Network exploitable

Access Complexity: Low

Authentication: Not required to exploit

Impact Type: Allows unauthorized disclosure of information; Allows unauthorized modification; Allows disruption of service

References to Advisories, Solutions, and Tools

By selecting these links, you will be leaving NIST webspace. We have provided these links to other web sites because they may have information that would be of interest to you. No inferences should be drawn on account of other sites being referenced, or not, from this page. There may be other web sites that are more appropriate for your purpose. NIST does not necessarily endorse the views expressed, or concur with the facts presented on these sites. Further, NIST does not endorse any commercial products that may be mentioned on these sites. Please address comments about this page to nvd@nist.gov.

External Source: CONFIRM

Name: http://bugs.python.org/issue20246

Type: Patch Information

Hyperlink: http://bugs.python.org/issue20246

External Source: MISC

Name: https://www.trustedsec.com/february-2014/python-remote-code-execution-socket-recvfrom_into/

Hyperlink: https://www.trustedsec.com/february-2014/python-remote-code-execution-socket-recvfrom_into/

Defender

The defender can change the structure of:

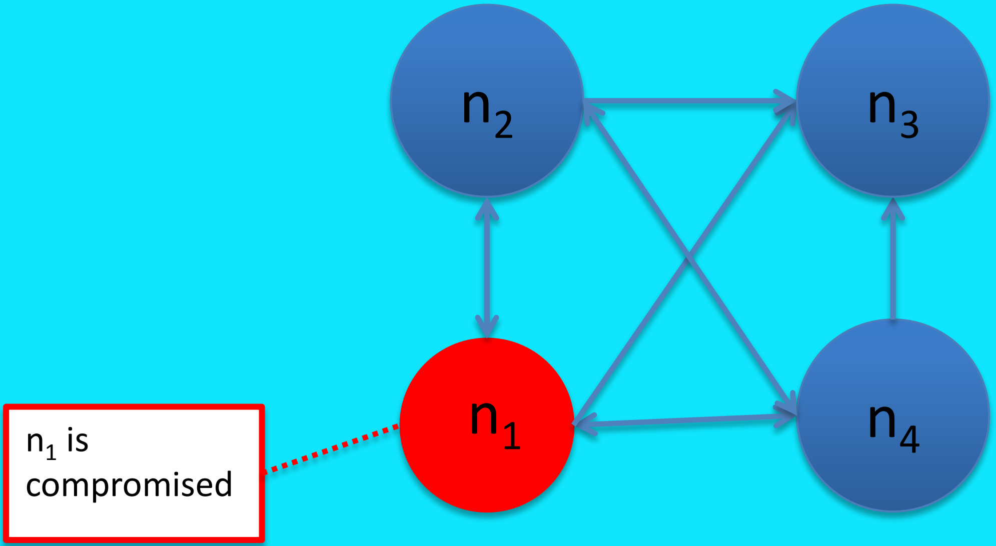
- Enterprise network
  - Add new honey hosts
- System vulnerability dependency graphs
  - Add new honey vulnerability
  - Patching vulnerabilities, i.e. removing vulnerabilities
  - Deactivate software, i.e. remove all the vulnerabilities related to a specific software

Enterprise Network

Definition 1 (Enterprise Network). *An* enterprise network EN *is a 3-tuple* $\mathsf{EN} = (N, E, \mathsf{compr})$ *where:*

1. N *is a set of nodes;*
2. $E \subseteq N \times N$ *is a set of edges;*
3. $\mathsf{compr} \subseteq N$ *is the set of compromised nodes.*

We define an algebra of operations on enterprise networks.



n_1 is compromised

Enterprise Networks: Union Operator

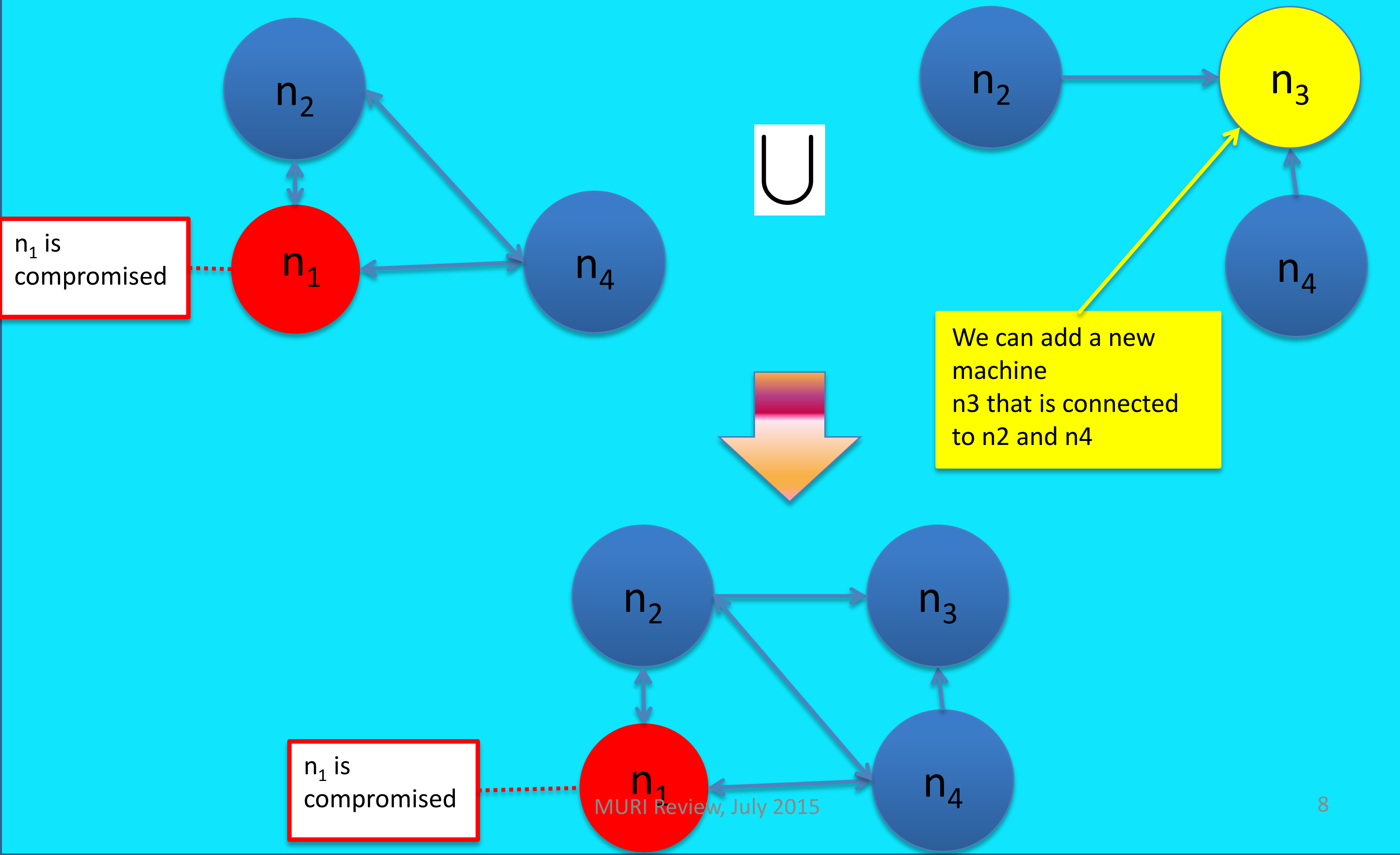
Definition 2 (Union of Enterprise Network). *Given* $SEN = \{EN^1, \ldots, EN^h\}$ *of enterprise networks, s.t.* $\mathsf{EN}_i = (N^i, E^i, \mathsf{compr}^i)$, *the resulting Enterprise Network obtained by the union operator is*

$$\mathsf{EN} = (N, E, \mathsf{compr}) = \bigcup_{i=1}^{h} EN^i$$

where

- $N = \bigcup_{i=1}^{h} V^i$;
- $E = \{(n_1, n_2) | n_1, n_2 \in N, (n_1, n_2) \in E^i, i \in \{1, \ldots, h\}\}$;
- $\mathsf{compr} = \bigcup_{i=1}^{h} \mathsf{compr}^i$.

Enterprise Networks: Union Operator



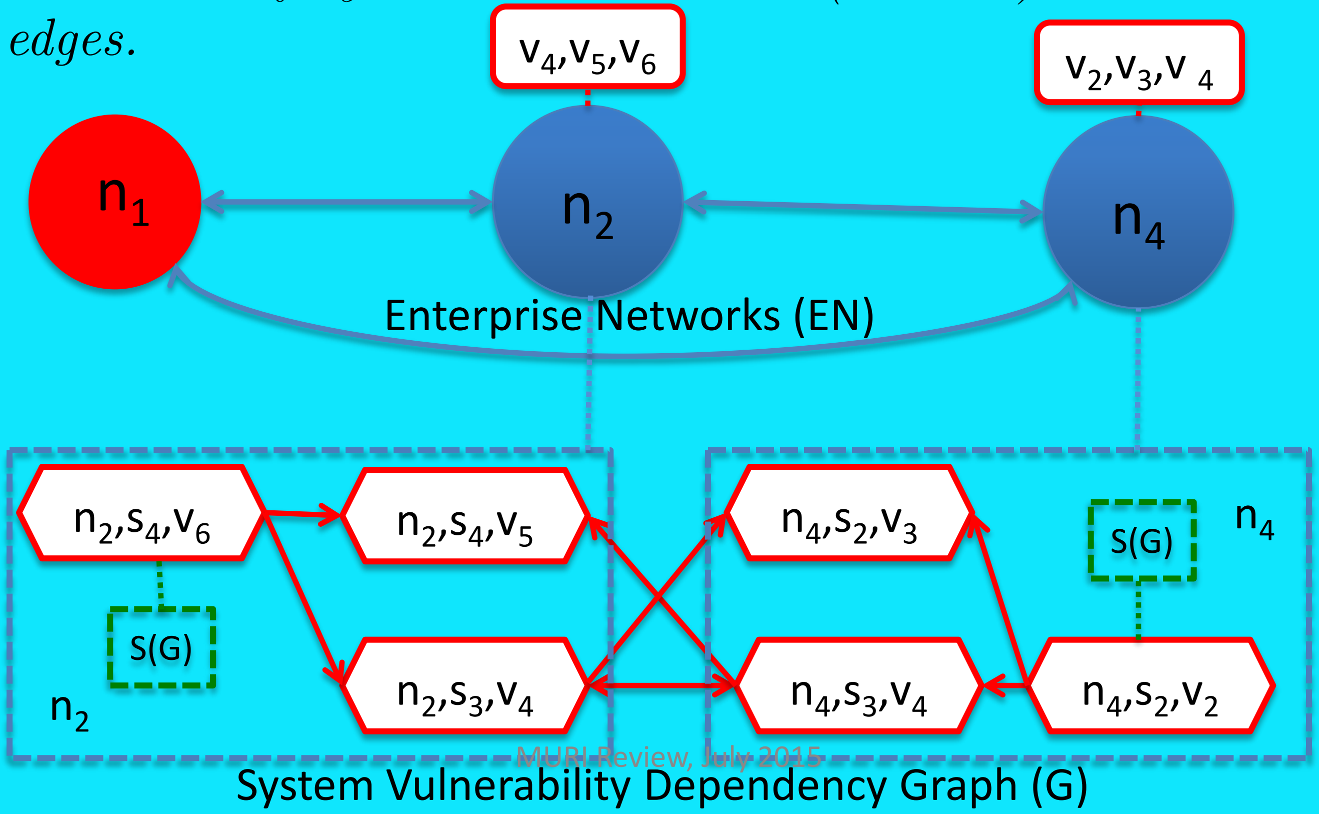
n_1 is compromised

We can add a new machine
n3 that is connected to n2 and n4

n_1 is compromised

System Vulnerability Dependency Graphs (SVDGs)

Definition 3 (System Vulnerability Dependency Graph). *Given an enterprise network* EN $= (N, E,$ compr$)$, *a* vulnerability dependency graph *(SVDG for short) is a directed graph $G = (\mathcal{SV}, Ev)$ where $\mathcal{SV} \subseteq \{(n, s, v) | n \in N, s \in \mathcal{S}, v \in \mathcal{V}\}$ is the set of system vulnerabilities (vertices) and $Ev \subseteq \mathcal{SV} \times \mathcal{SV}$ is the set of edges.*



SVDG Algebra: Union and Difference Operators

Definition 4 (Union). *Given a set* $SG = \{G^1, dots, G^h\}$ *of vulnerability dependency graphs s.t.* $G^i = (\mathcal{SV}^i, Ev^i)$ *the resulting graphs obtained by the union operator is*

$G = (\mathcal{SV}, Ev) = \bigcup_{i=1}^{h} G^i$ *where*

- $\mathcal{SV} = \bigcup_{i=1}^{h} \mathcal{SV}^i$;
- $Ev = \{(v_1, v_2) | v_1, v_2 \in \mathcal{SV}, (v_1, v_2) \in Ev^i, i \in \{1, \ldots, h\}\}$;

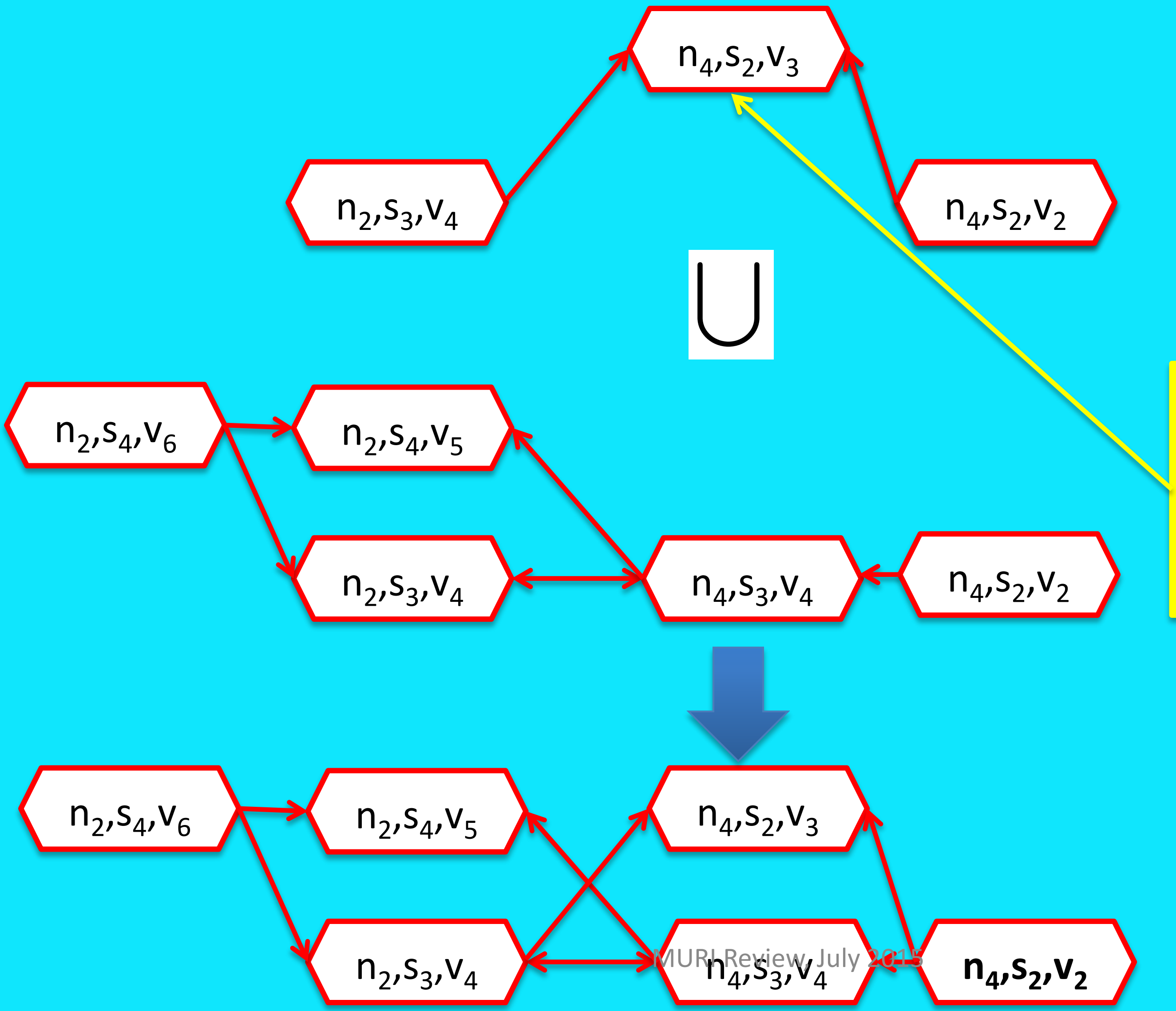
Defender can add "apparent" vulnerabilities.

Definition 5 (Difference). *Given two vulnerability dependency graphs* $G_1 = (\mathcal{SV}^1, Ev^1)$ *and* $G_2 = (\mathcal{SV}^2, Ev^2)$*, the difference result is* $G = (\mathcal{SV}, Ev) = G_1 \setminus G2$ *s.t.*

- $\mathcal{SV} = \mathcal{SV}^1 \setminus \mathcal{SV}^2$;
- $Ev = \{(v_1, v_2) | v_1, v_2 \in \mathcal{SV}, (v_1, v_2) \in Ev^1\} \setminus E^2$;

Defender can remove vulnerabilities, e.g. by uninstalling relevant software or by patching

SVDG Algebra: Union Operator



n_4,s_2,v_3

n_2,s_3,v_4

n_4,s_2,v_2

$\cup$

n_2,s_4,v_6

n_2,s_4,v_5

n_2,s_3,v_4

n_4,s_3,v_4

n_4,s_2,v_2

n_2,s_4,v_6

n_2,s_4,v_5

n_4,s_2,v_3

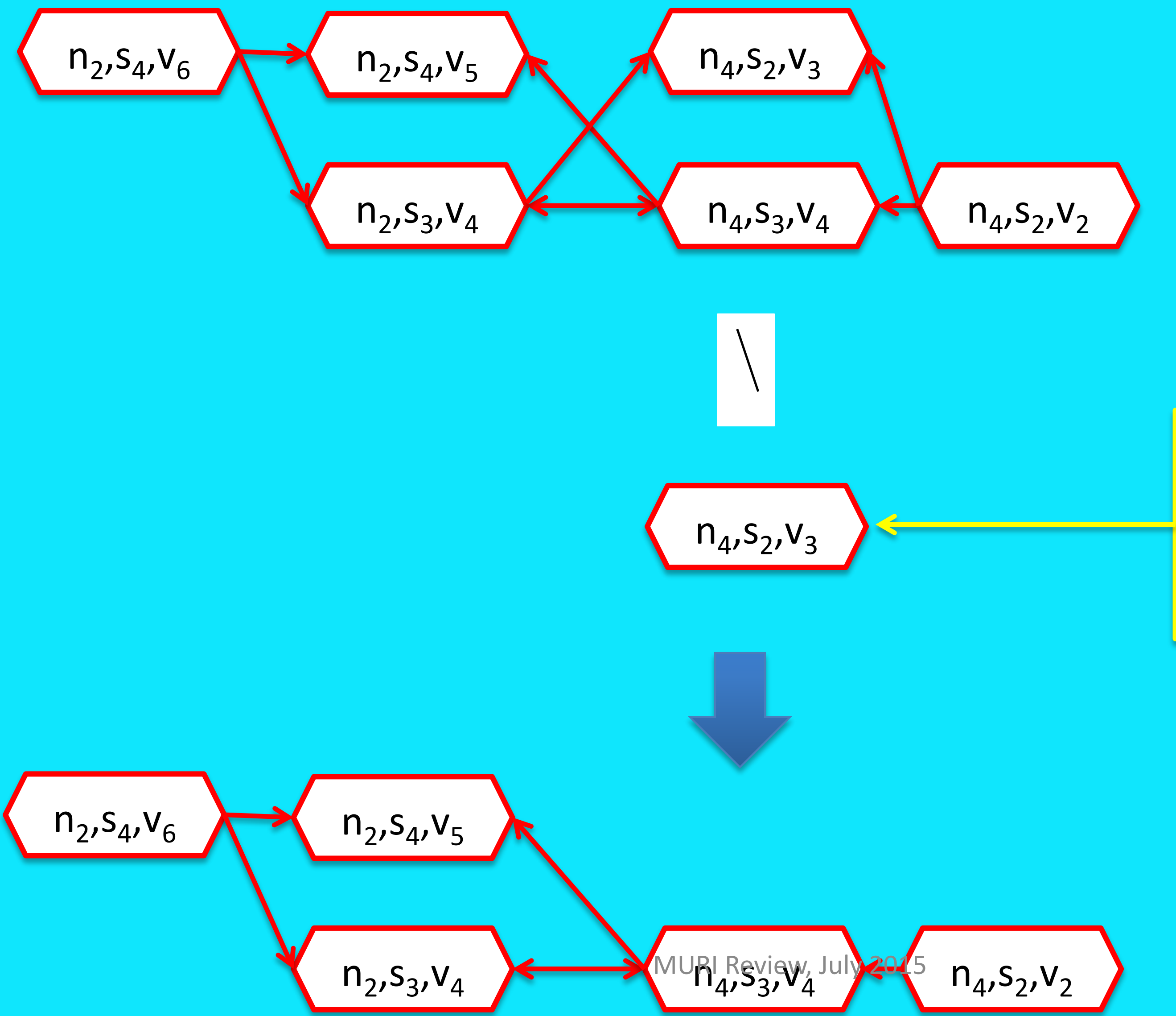
n_2,s_3,v_4

n_4,s_3,v_4

$\mathbf{n_4,s_2,v_2}$

We can add one or more “apparent” new vulnerabilities along with relevant connections

SVDG Algebra: Difference Operator



We can patch a vulnerability or deactivate a software module

Defender Strategy

Definition 12 (Add Enterprise Universe). *The Add Enterprise Universe is* $AB = \{ab_1, \ldots, ab_h\}$ *where* $ab_i = (\mathsf{EN}^i, G^i)$ *is a pair composed by an enterprise networks* EN^i *and a vulnerability dependency graph* G^i*.*

Definition 13 (Remove SVDG Universe). *The Remove SVDG Universe is* $RB = \{b_1, \ldots, b_h\}$ *where* $rb_i = (G^i)$ *is a system vulnerability dependency graph.*

Definition 14 (Defender Strategy). *The strategy of the defender* $\delta = (A, R)$ *is a pair of two sets* A *and* R *s.t.* $A \subseteq AB$ *and* $R \subseteq RB$*.*

Defender can add honey nodes/vulnerabilities to the network

Defender can remove vulnerabilities via patching/deactivation

Defender strategy: set of actions (add,remove)

Cost of a Defender Strategy

Given a defender strategy $\delta = (A, R)$ we define several measures about cost and productivity

Definition 14 (Node Cost).

$$ncost(A) = \sum_{n \in N', (N',) = \bigcap_{(\mathsf{EN}^i,) \in A} \mathsf{EN}^i} costD(n)$$

Cost of adding honey nodes

Definition 15 (Patch Cost).

$$pcost(R) = \sum_{G^i \in R} costD(G^i)$$

Cost of patching

Definition 16 (Productivity Cost).

$$prcost(R) = \sum_{G^i \in R} pcost(G^i)$$

Cost of unhappiness caused by deactivating software

Defender Strategy Configuration Result

Definition 15. *Given a defender strategy* $\delta = (A, R)$*, the resulting enterprise network, system vulnerability dependency graph, and honey set are:*

$$\mathsf{EN}(A, R) = \mathsf{EN} \cup \Big(\bigcup_{(\mathsf{EN}^{i}{}_{,)} \in A} \mathsf{EN}^{i} \Big) \tag{1}$$

$$G(A, R) = \Big(G \setminus \bigcup_{G_i \in R} G_i\Big) \cup \Big(\bigcup_{(,G^{i}) \in A} G_i \Big) \tag{2}$$

$$honey = \bigcup_{(\_,(\mathcal{SV}\ ,\_)) \in A} \mathcal{SV} \tag{3}$$

The Attacker Side

- Two types of attack actions
  - Exploiting vulnerabilities
  - Scanning nodes
- Attacker actions change his state (e.g. his knowledge about the enterprise network)
  - State model
  - In theory, a “rational” attacker will maximize his utility.
  - In practice and in our experiments, we allow sub-rational actors.

Attacker State Model

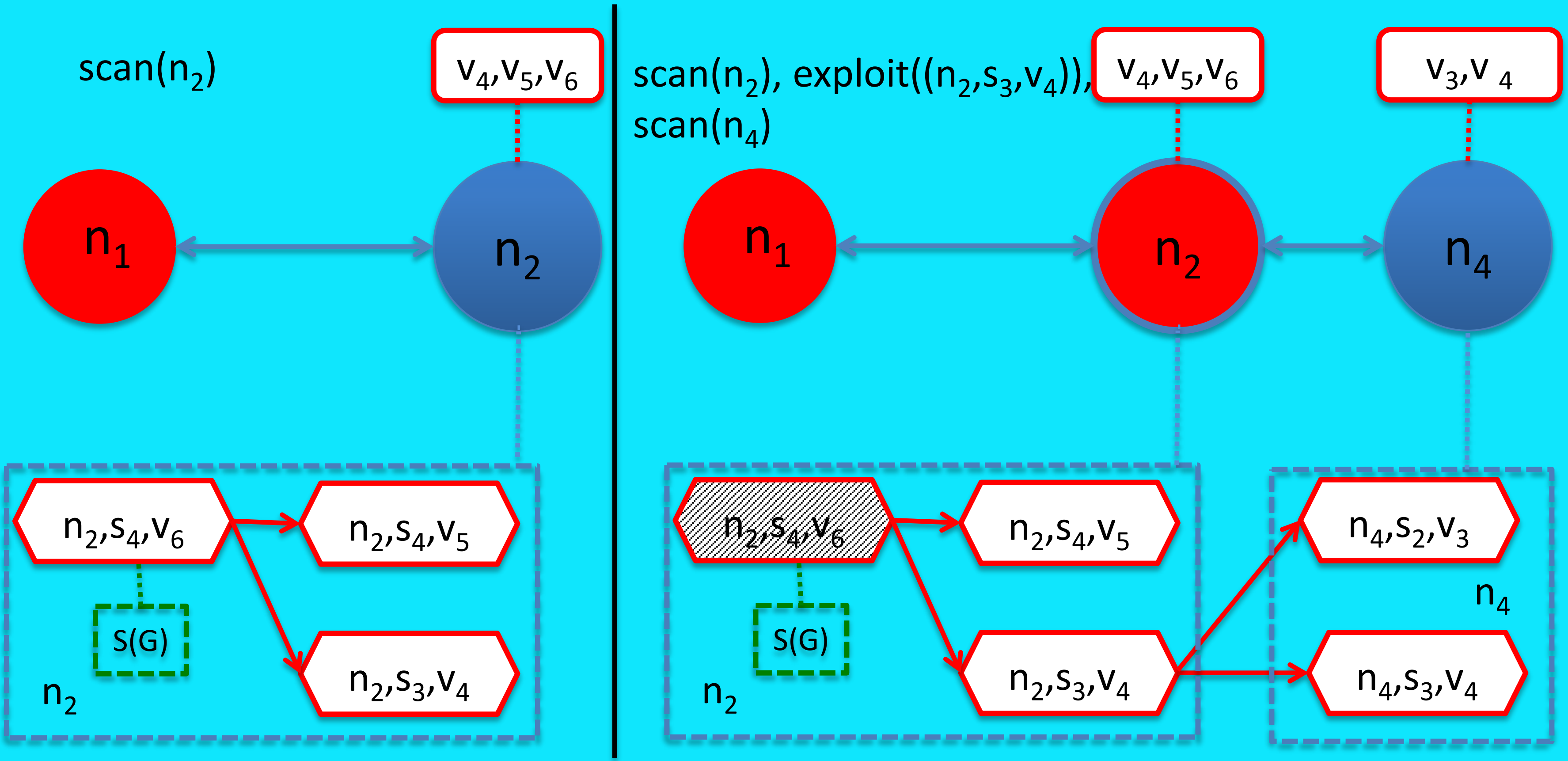
Definition 6 (Attacker Actions). *Given an enterprise network* $\mathsf{EN} = (N, E, \mathsf{honey}, \mathsf{compr})$ *and a system vulnerability dependency graph* $G = (\mathcal{SV}, Ev)$*, the set of all possible actions of an attacker is* $A(\mathsf{EN}, G) = SC(\mathsf{EN}) \cup EXP(G)$ *where*

- $SC(\mathsf{EN}) = \{scan(n) | n \in N\}$ *is the set of all possible scan actions;*
- $EXP(G) = \{exploit(v) | v \in V\}$ *is the set of all possible exploit actions.*

Attacker state $(\boldsymbol{Acts_i}, \boldsymbol{EN_i}, \boldsymbol{G_i})$ **consists of three things:**

- The history of actions the attacker took to get to this state
- The enterprise network the attacker knows through these actions
- The SVDG that the attacker knows through his actions

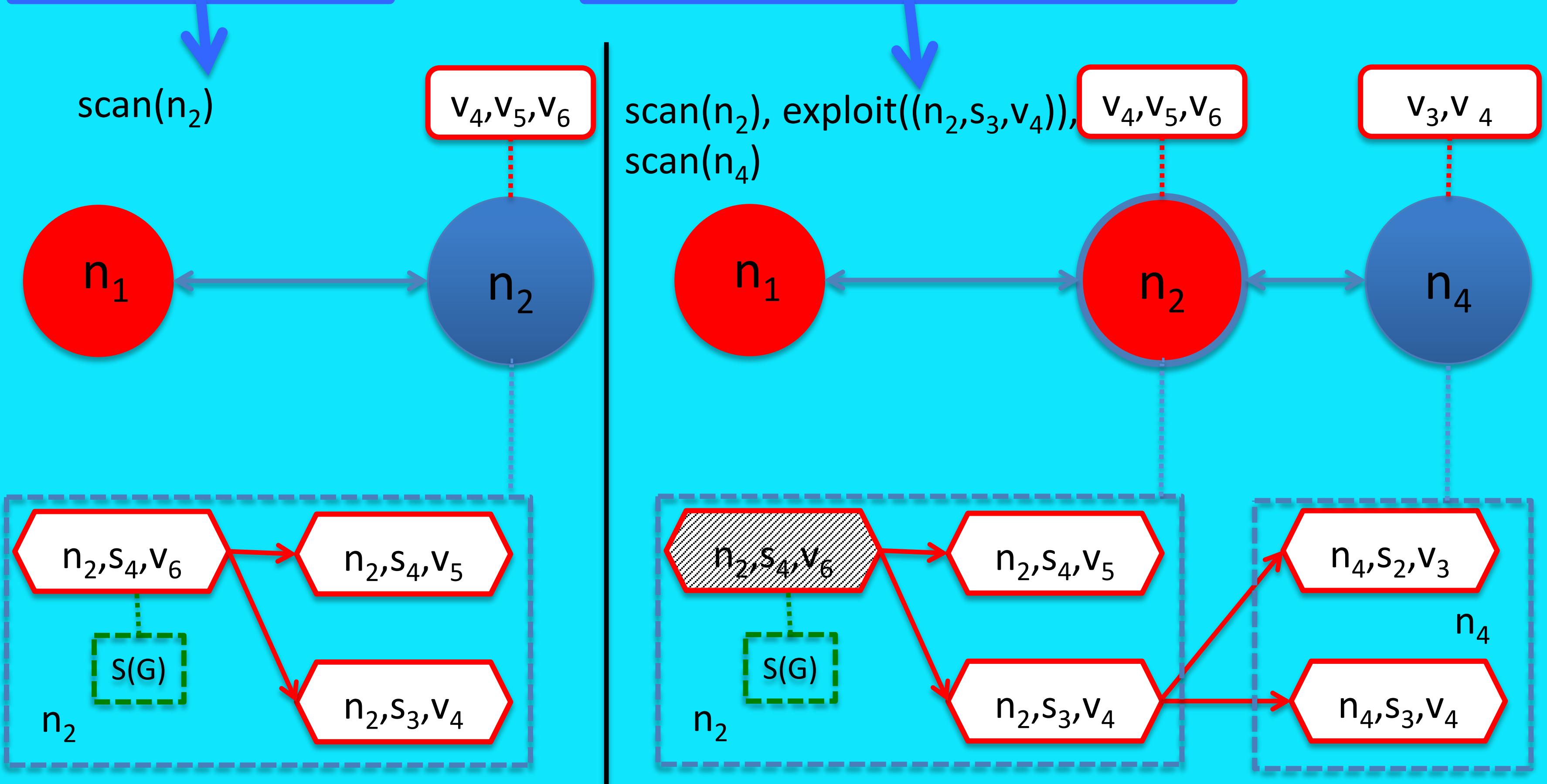
Attacker State Model



Attacker State Model

State obtained after $scan(n_2)$

State obtained after $scan(n_2)$, $exploit((n_2,s_3,v_4))$, $scan(n_4)$



Attacker State Model

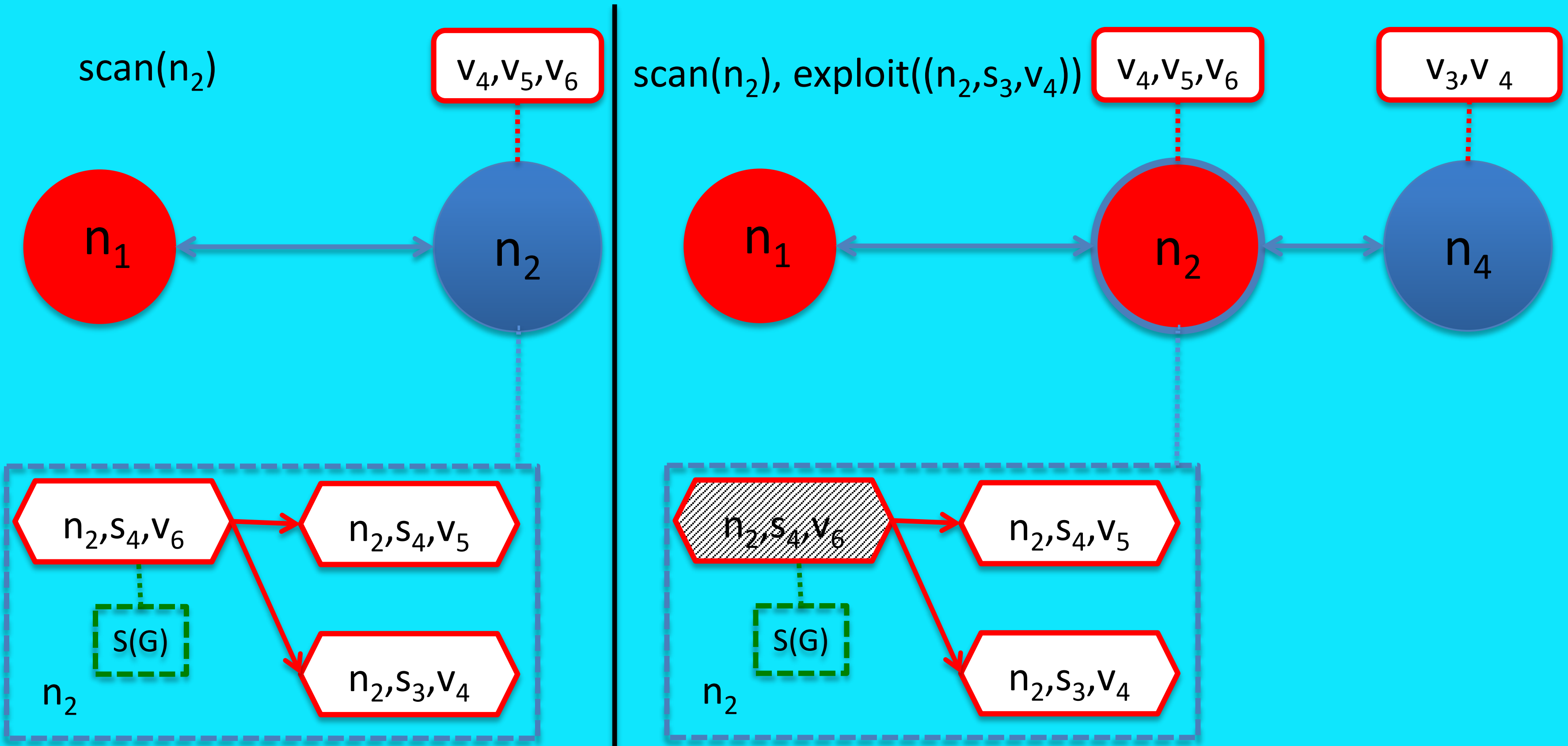
Definition 9 (Transition State). *Given an attacker state* $s_i = (Acts_i, (N_i, E_i, \mathsf{honey}_i, \mathsf{compr}_i), (\mathcal{SV}_i, Ev_i))$ *and an action* $a \in VA(s_i)$*, the transition state* $tr(s_i, a)$ *is equal to* $s_{i+1} = (Acts_{i+1}, \mathsf{EN}_{i+1}, G_{i+1})$ *where*

- *if* $a = scan(n)$ *then* $Acts_{i+1} = Acts_i \cup \{scan(n)\}$, $\mathsf{EN}_{i+1} = (N_i, E_i, \mathsf{compr}_i)$ *and* $G_{i+1} = (\mathcal{SV}_{i+1} = \mathcal{SV}_i \cup \{(n,s,v) | (n,s,v) \in \mathcal{SV}\}, \{(v,v') | v, v' \in \mathcal{SV}_{i+1}, (v,v') \in E\})$

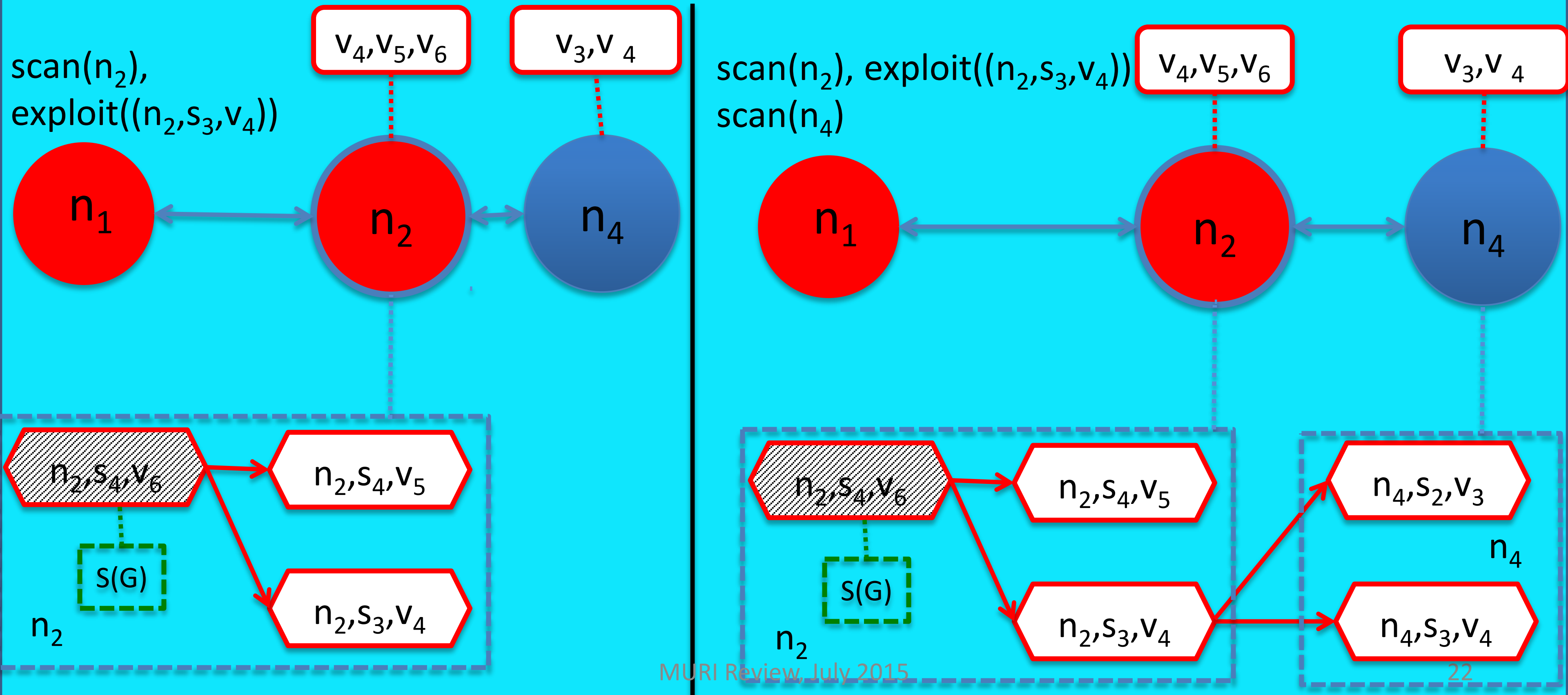
- *if* $a = exploit((n,s,v))$ *then* $Acts_{i+1} = Acts_i \cup \{exploit((n,s,v))\}$, $G_{i+1} = G_i$ *and*

$$\mathsf{EN}_{i+1} = (N_i \cup \{n' | (n,n') \in E\}, E_i \cup \{\{(n,n') | (n,n') \in E\}, \mathsf{compr}_i)$$

Transition State exploit((n_2,s_3,v_4))



Transition State
scan(n_4)



scan(n_2),
exploit((n_2,s_3,v_4))

scan(n_2), exploit((n_2,s_3,v_4))
scan(n_4)

Valid Attacker Strategy

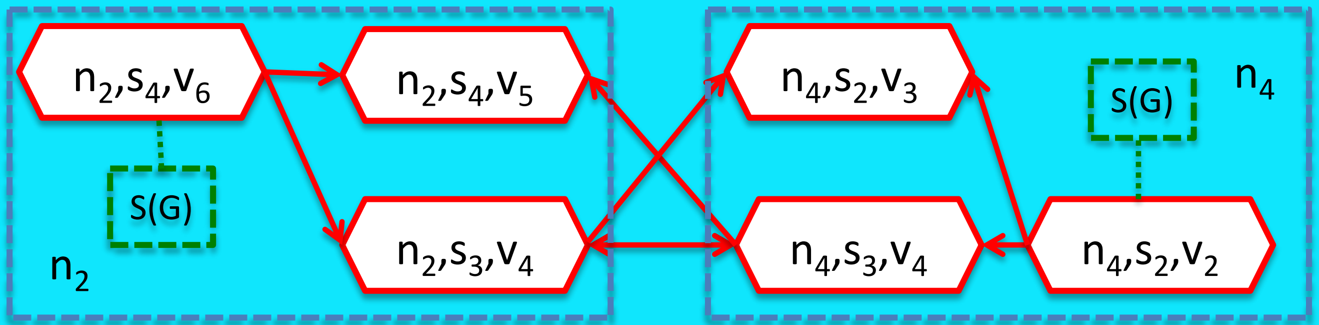
Definition 10 (Valid Attacker Strategy). *Given an enterprise network* $\mathsf{EN} = (N, E, \mathsf{honey}, \mathsf{compr})$, *a system vulnerability dependency graph* $G = (\mathcal{SV}, Ev)$ *and threshold cost* $\hat{c}$, *a sequence* $as =< a_1, \ldots, a_m >$ *is a valid attacker strategy if there exists an associated sequence of attacker state* $< s_0, s_1, \ldots, s_m >$ *s.t.*

- $\forall i \in \{1, \ldots, m\} : \; a_i \in VA(s_{i-1})$
- $\forall i \in \{1, \ldots, m\} : \; s_i = tr(s_{i-1}, a_i)$
- $\sum_{i=1}^{m} costA(a_i) \leq \hat{c}$
- $\forall a \in VA(s_m) : \; costA(a) + \sum_{i=1}^{m} costA(a_i) > \hat{c}$

Cost of attack (e.g. probability of detection) should be below a threshold

Adding another valid attack should cause the threshold to be exceeded.

Valid Attacker Strategy



System Vulnerability Dependency Graph (G)

1) n_2,s_4,v_6 n_2,s_3,v_4 n_2,s_4,v_5 n_4,s_2,v_3

2) n_4,s_2,v_2 n_4,s_2,v_3 ~~n_2,s_4,v_5~~

3) n_4,s_2,v_2 ~~n_2,s_4,v_5~~ n_4,s_3,v_4

Utility Driven Method

Given an attacker strategy $as =< a_1, \ldots, a_m >\in AS(\mathsf{EN}, G, \hat{c})$ and its state sequence $ss(as) =< s_0, s_1, \ldots, s_m >$ the attacker strategy probability function is

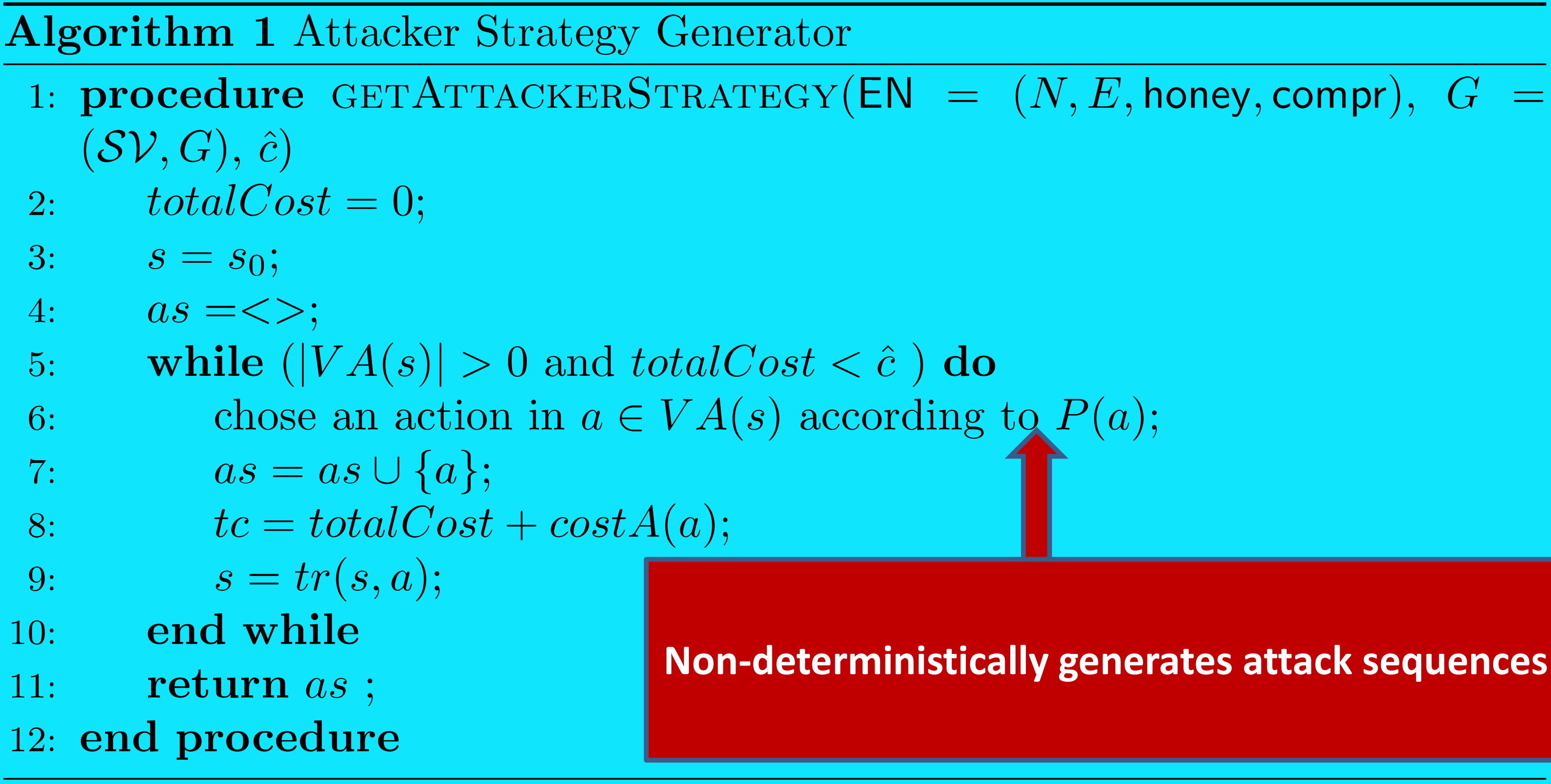
$$P(as) = \prod_{i=1}^{m} Pr(s_{i-1}, a_i)$$

where $Pr(s, a)$ is defined in the following way:

$$Pr_1(s,a) \quad = \quad \frac{utilA(a, VA(s))}{\sum_{a' \in VA(s)} utilA(a', VA(s))}$$

Probability of a specific attacker strategy is based on relative utility

Utility Driven Method



Algorithm 1 Attacker Strategy Generator

```
procedure GETATTACKERSTRATEGY($\mathsf{EN} = (N, E, \mathsf{honey}, \mathsf{compr}),\ G = (\mathcal{SV}, G),\ \hat{c}$)
    $totalCost = 0$;
    $s = s_0$;
    $as = <>$;
    while ($|VA(s)| > 0$ and $totalCost < \hat{c}$ ) do
        chose an action in $a \in VA(s)$ according to $P(a)$;
        $as = as \cup \{a\}$;
        $tc = totalCost + costA(a)$;
        $s = tr(s, a)$;
    end while
    return $as$ ;
end procedure
```

Non-deterministically generates attack sequences

Vulnerability & Data Impact

Definition 17 (Vulnerability impact). *Given a function impact associating to each system vulnerability in the system an impact value, we define the* vulnerability impact of an attacker strategy *as in two ways:*

$$vimpactDA1(as) = \max_{exploit(v) \in as, v \notin honey} impact(v)$$

$$vimpactDA2(as) = \sum_{exploit(v) \in as, v \notin honey} impact(v)$$

Definition 18 (Data impact). *Given a function data associating to each vulnerability the set of data disclosed, and a function impactData providing the impact for each disclosed data, we define the* data impact of an attacker strategy *as in two ways:*

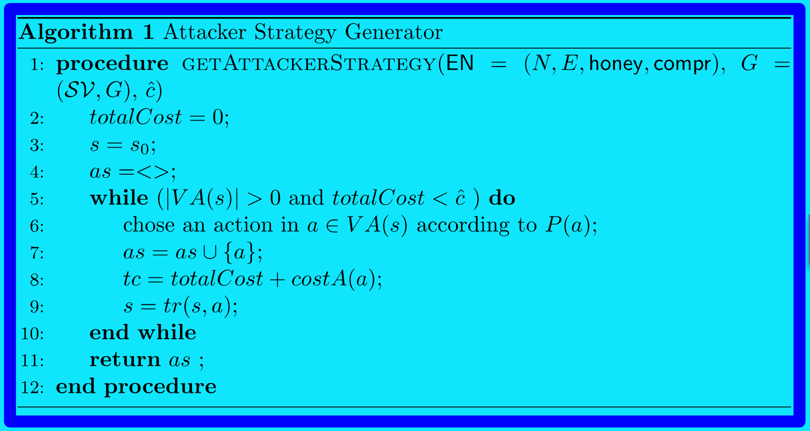
$$dimpactDA1(as) = \max_{d \in data(v), exploit(v) \in as, v \notin honey} impactData(d)$$

$$dimpactDA2(as) = \sum_{d \in data(v), exploit(v) \in as, v \notin honey} impactData(d)$$

Expected Impacts

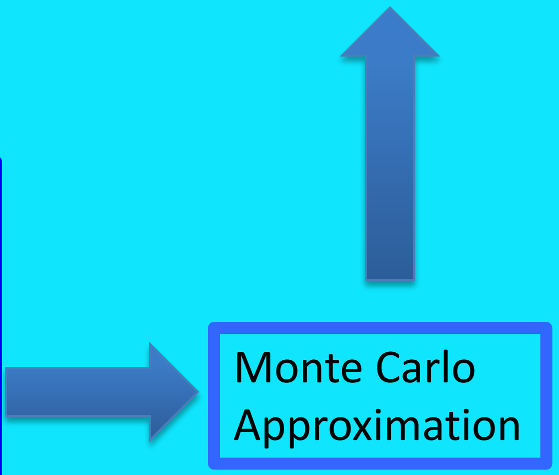
Definition 19 (Expected Impact)**.**

$$E[impactDA, \mathsf{EN}, G, \hat{c}] = \sum_{as \in AS(\mathsf{EN}, G, \hat{c})} P(as) \cdot impactDA(as)$$



Algorithm 1 Attacker Strategy Generator

```
procedure GETATTACKERSTRATEGY($\mathsf{EN} = (N, E, \mathsf{honey}, \mathsf{compr})$, $G = (\mathcal{SV}, G)$, $\hat{c}$)
    $totalCost = 0$;
    $s = s_0$;
    $as = <>$;
    while ($|VA(s)| > 0$ and $totalCost < \hat{c}$) do
        chose an action in $a \in VA(s)$ according to $P(a)$;
        $as = as \cup \{a\}$;
        $tc = totalCost + costA(a)$;
        $s = tr(s, a)$;
    end while
    return $as$ ;
end procedure
```



Monte Carlo Approximation

Pareto Optimal Defender strategy

$$\delta^* = (A^*, R^*) \in \underset{A \subseteq AB, R \subseteq RB}{\arg\min} \{$$

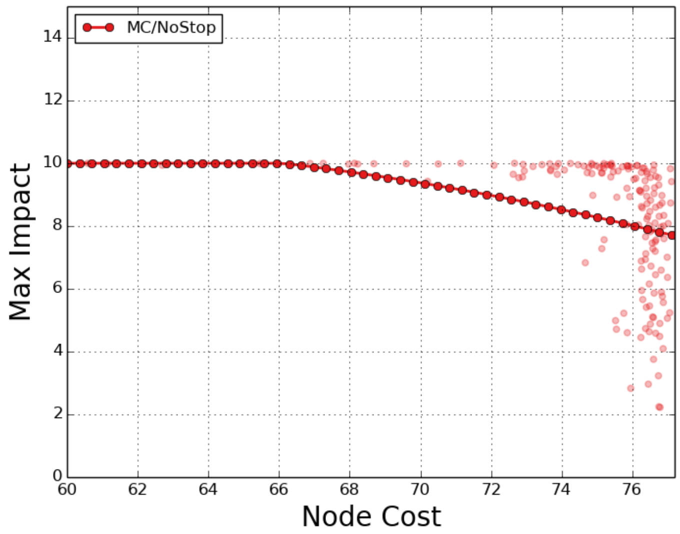
$$E[vimpactDA1, \mathsf{EN}(A,R), G(A,R), \hat{c}],$$
$$E[vimpactDA2, \mathsf{EN}(A,R), G(A,R), \hat{c}],$$
$$E[dimpactDA1, \mathsf{EN}(A,R), G(A,R), \hat{c}],$$
$$E[dimpactDA2, \mathsf{EN}(A,R), G(A,R), \hat{c}],$$
$$ncost(A),$$
$$pcost(R),$$
$$prcost(R)$$
$$\}$$

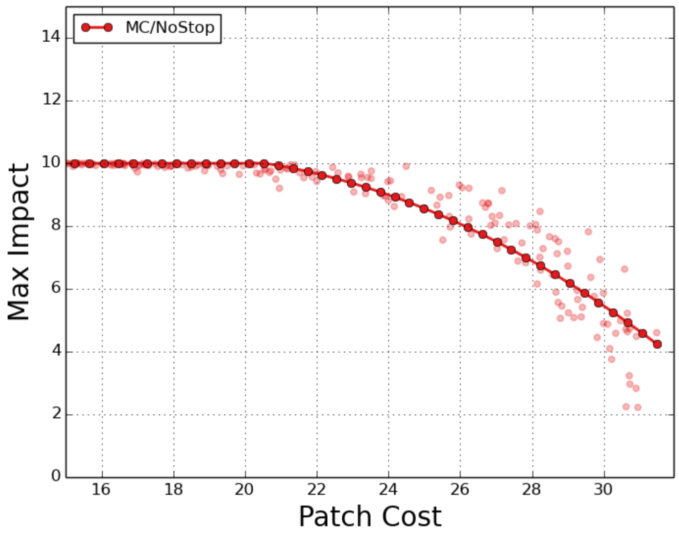
Compute Pareto Frontier

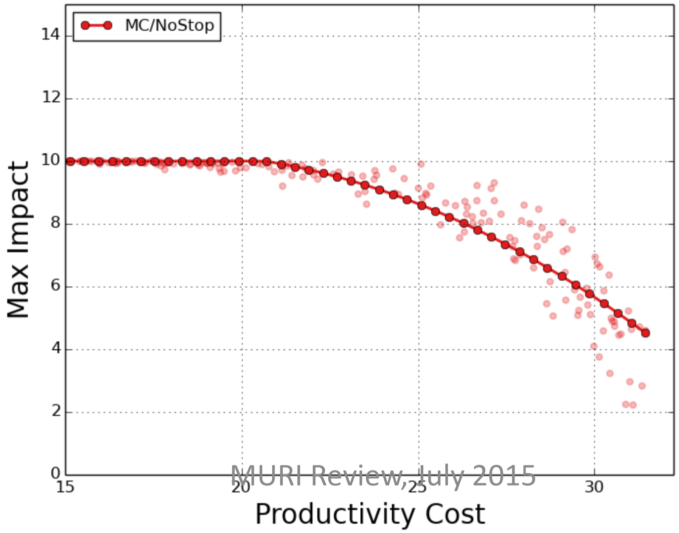
Theorem. Finding optimal defender strategy is NP-hard.

Developed heuristic algorithm based on genetic algorithms with reinforcement learning for the problem.

Experiment: Attacker continues after detection

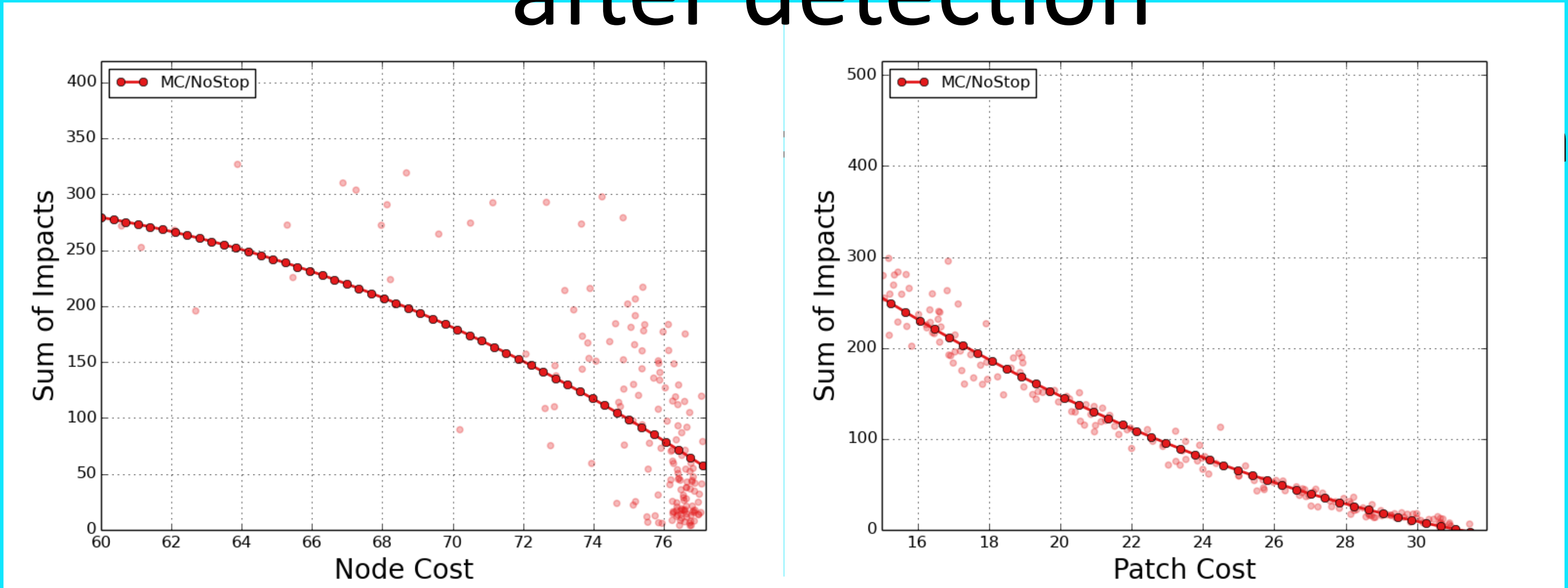




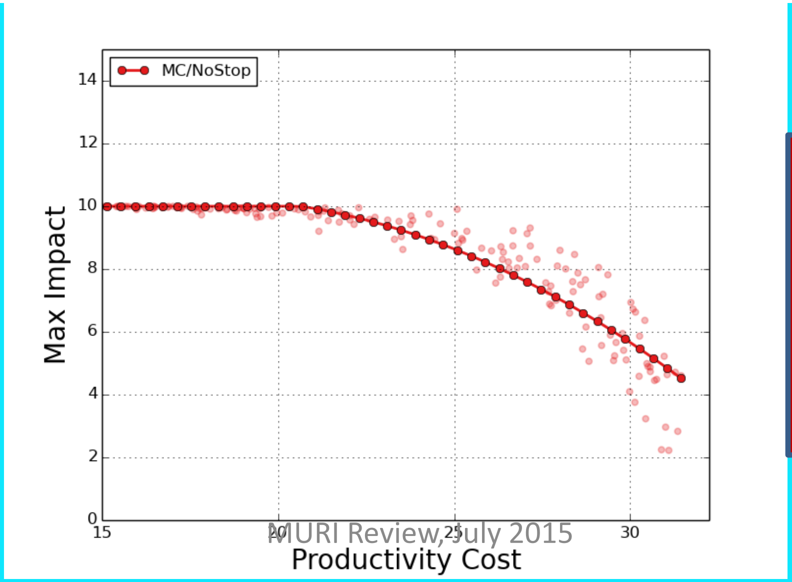


When the defender incurs additional costs (and allows the attacker to continue even after detection, impact goes down)

Experiment: Attack continues after detection



t



When the defender incurs additional costs (and allows the attacker to continue even after detection, impact goes down)

Defender Stops Attacker Immediately

The defender instantaneously stops the attacker each time he choses a honey vulnerability/node

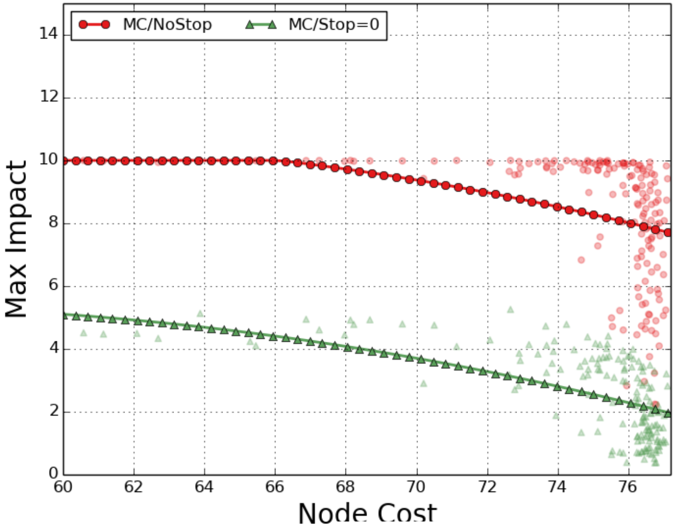
– changing of the expected impact

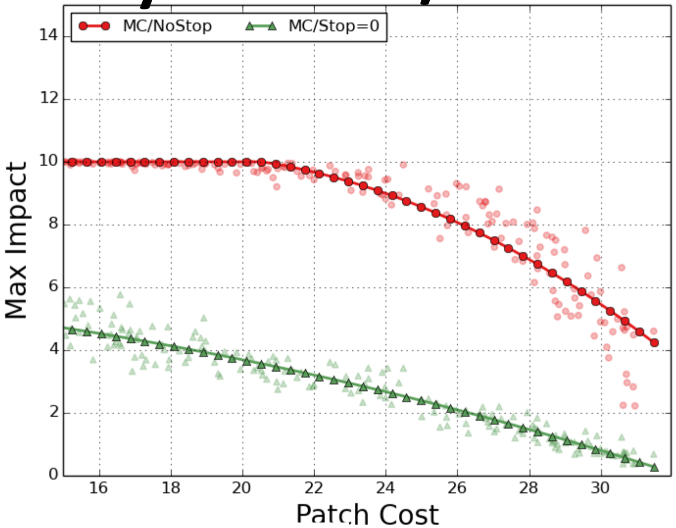
Definition 20 (Stop Operator). *Given an attacker strategy as* $< a_1, \ldots, a_m >$, *the stop operator returns a sub subsequece* $stops(as) =< a_1, \ldots, a_h > s.t.\ h = \min_{a_i = exploit(v) \in as, v \in \mathsf{honey}} i.$

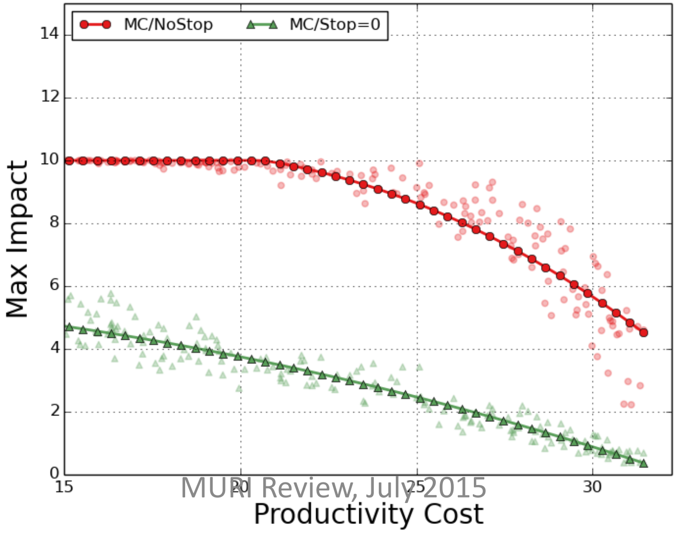
Definition 21 (Stop Expected Impact).

$$E[impactDA, \mathsf{EN}, G, \hat{c}] = \sum_{as \in AS(\mathsf{EN}, G, \hat{c})} P(as) \cdot impactDA(stops(as))$$

Experiment: Defender stops attacker when he visits a honey node/vuln.

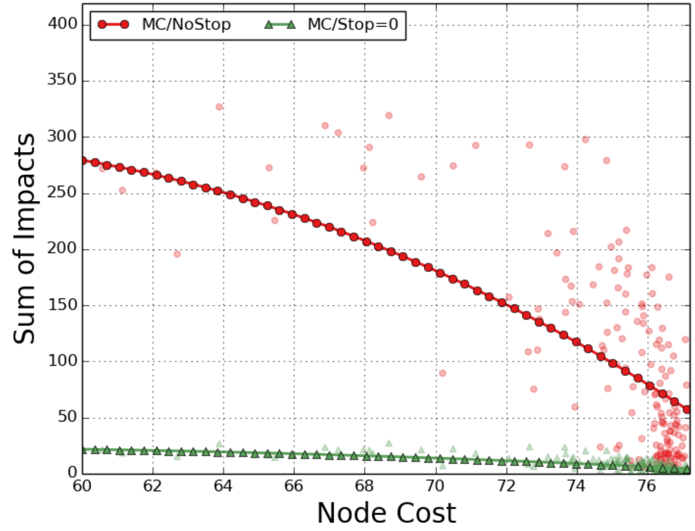


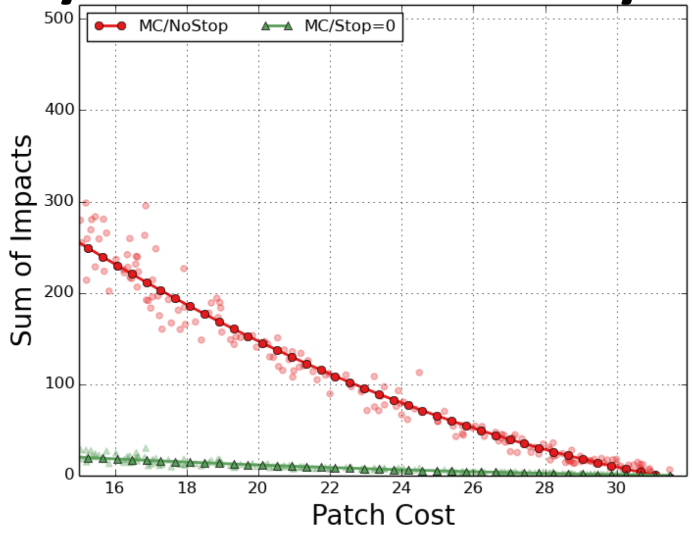


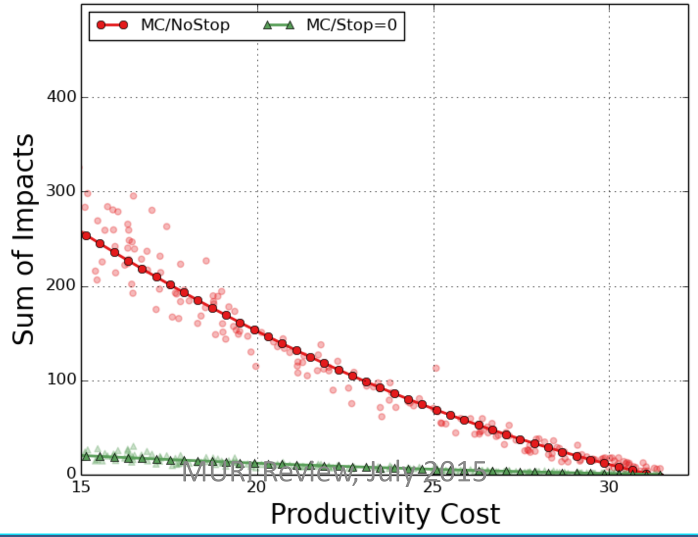


When the defender incurs additional costs (and stops the attacker immediately after detection), clearly we have better defense

Experiment: Defender Stops Attacker wben he visits honey vulnerability





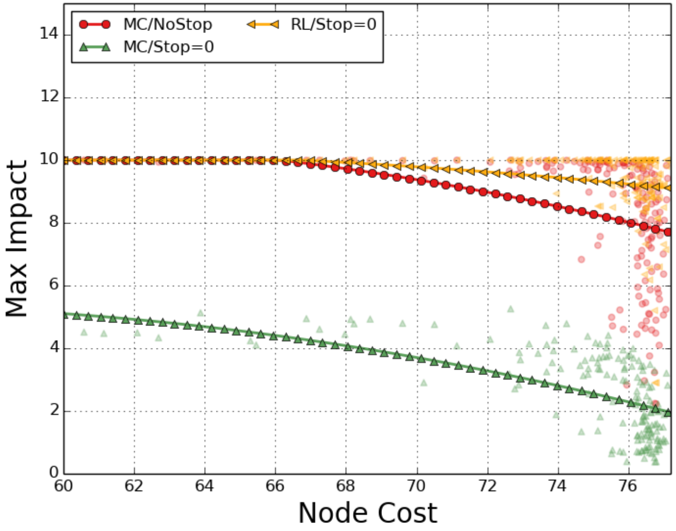


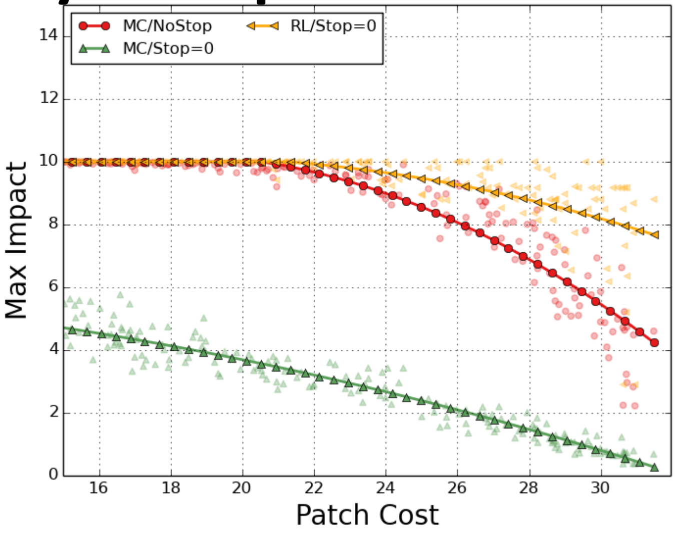
When the defender incurs additional costs (and stops the attacker immediately after detection), clearly we have better defense

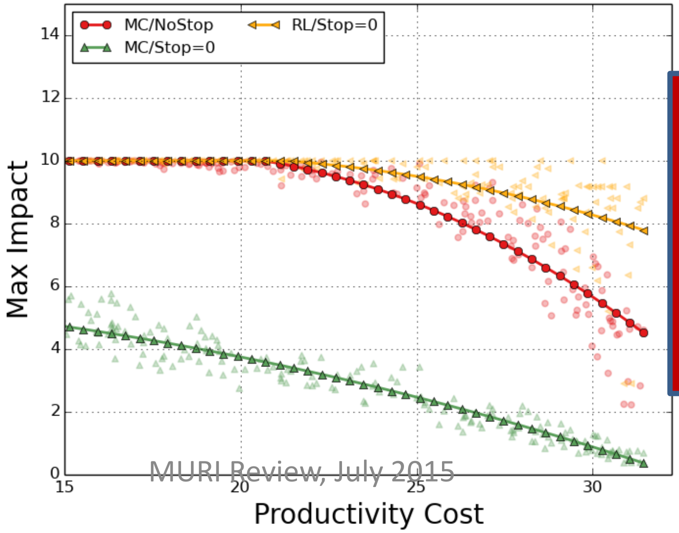
Attackers that Learn

- If the defender instantaneously stops the attacker when he visits a honey vulnerability, the attacker can learn the defender strategy and improve his strategy.
- We use the UCT (Reinforcement Learning) algorithm to simulate the attacker's ability to learn.
- For each vulnerability exploited, the attacker receives a reward equal to the impact of the vulnerability.
- If the attacker is stopped by the defender, the reward of the strategy as a whole is becomes zero.
- After reinforcing learning, our algorithm returns an optimal distribution (for the attacker) of the attacker strategies
- Recompute expected impact with this distribution.

Experiment: Attacker uses RL, Defender immediately stops attacks

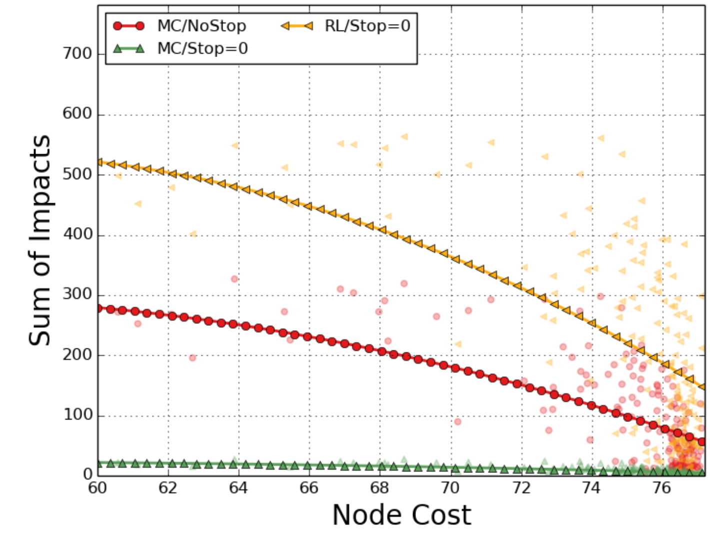


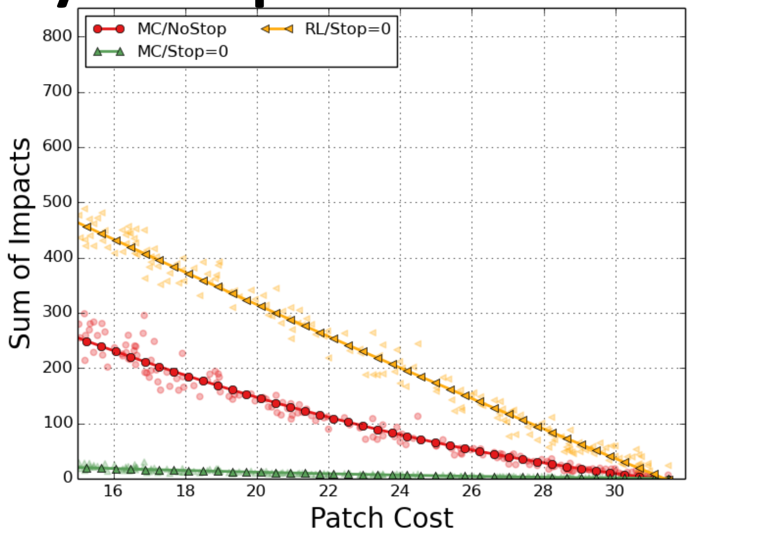


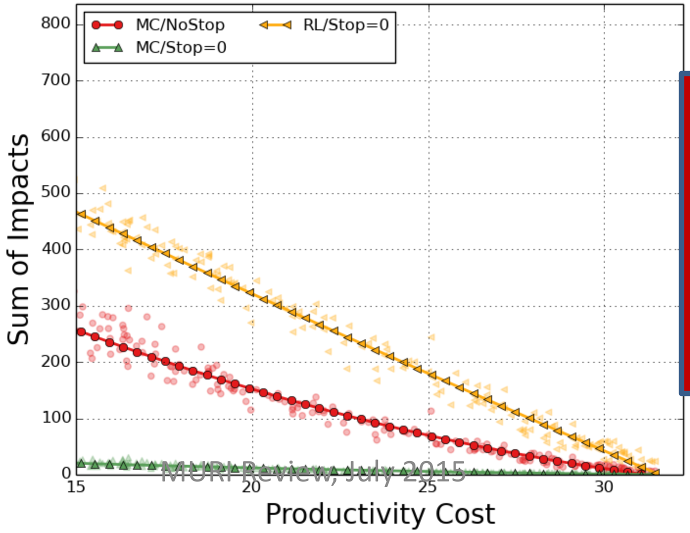


RL helps the attacker do better.

Experiment: Attacker uses RL, Defender immediately stops attacks





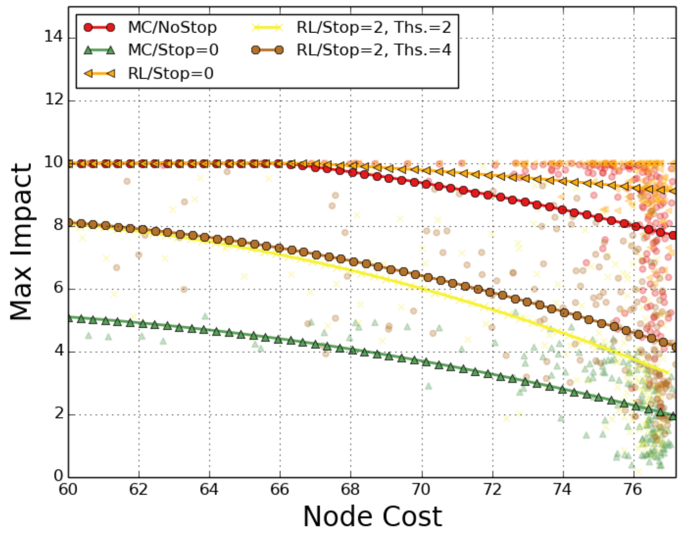


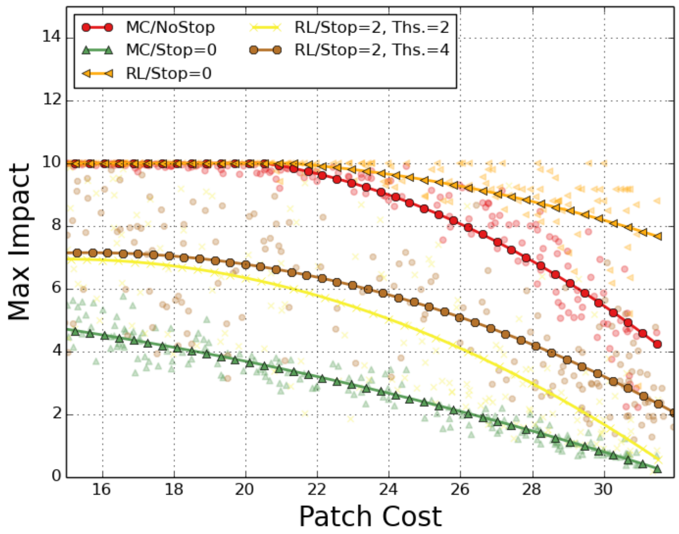
RL helps the attacker doe better.

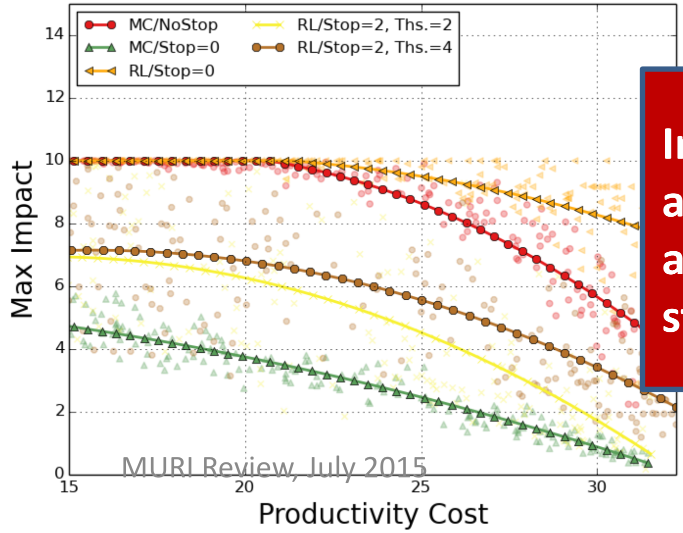
Delayed Stops

- Defender wants to increase uncertainty for the attacker.
- Defender wants to reduce attacker’s ability to learn.
- The defender stops an attacker (after he uses a honey vulnerability) only when he exploits a sufficiently dangerous vulnerability (*impact(v) >Ths*).

Experiment

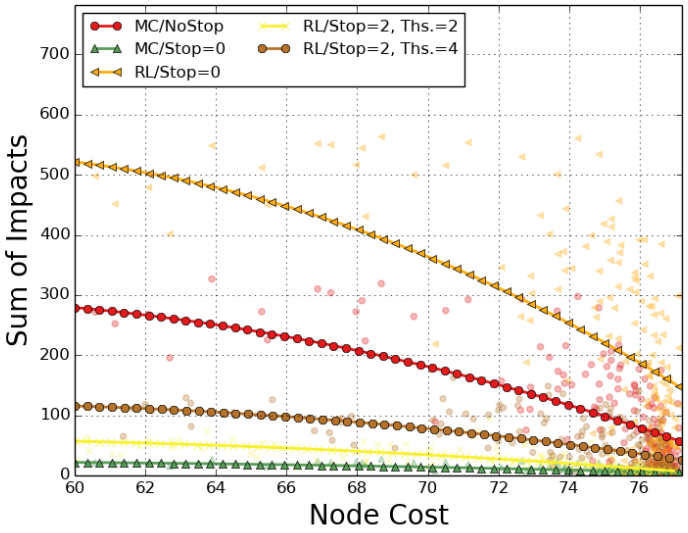


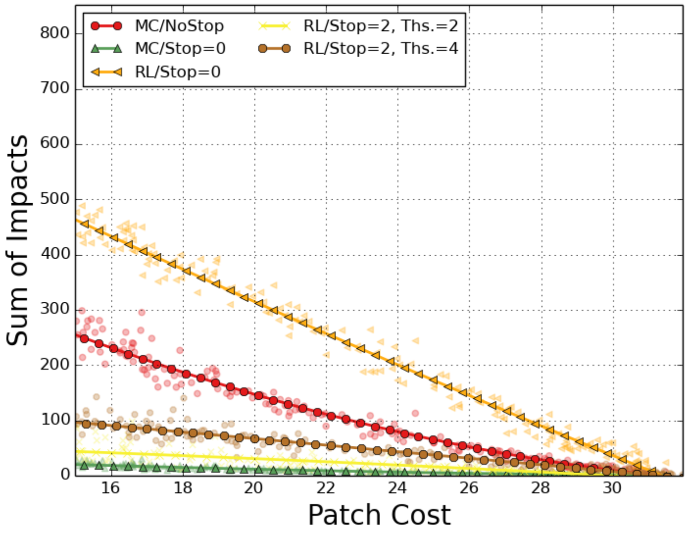


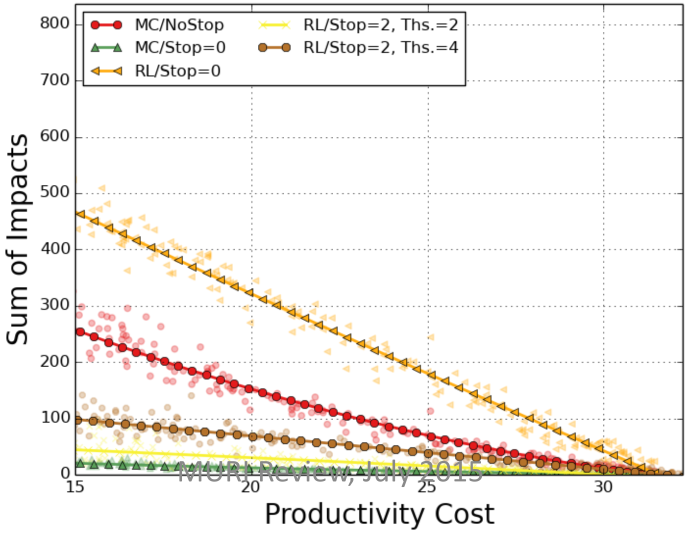


In order to target smart adversaries, delayed stops are better than immediate stops.

Experiment





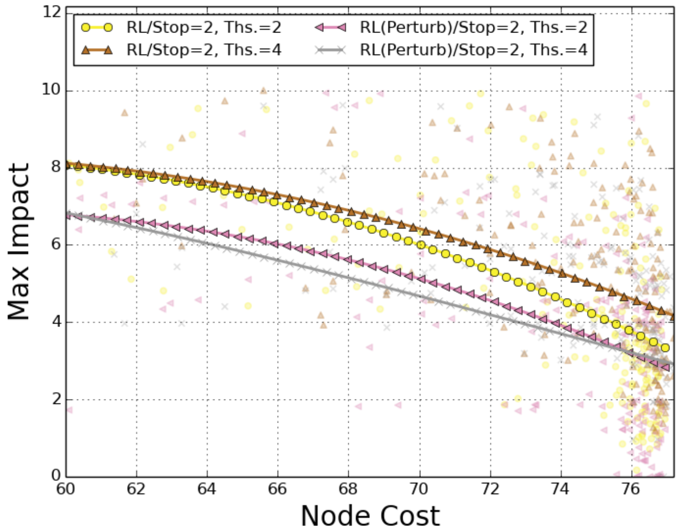


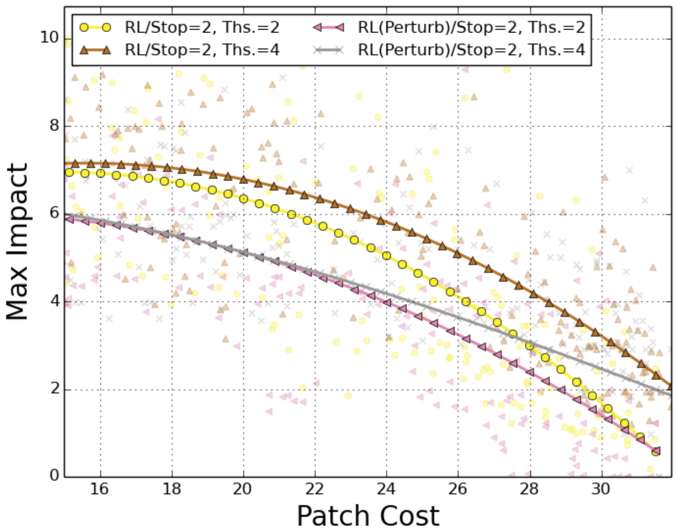
In order to target smart adversaries, delayed stops are better than immediate stops.

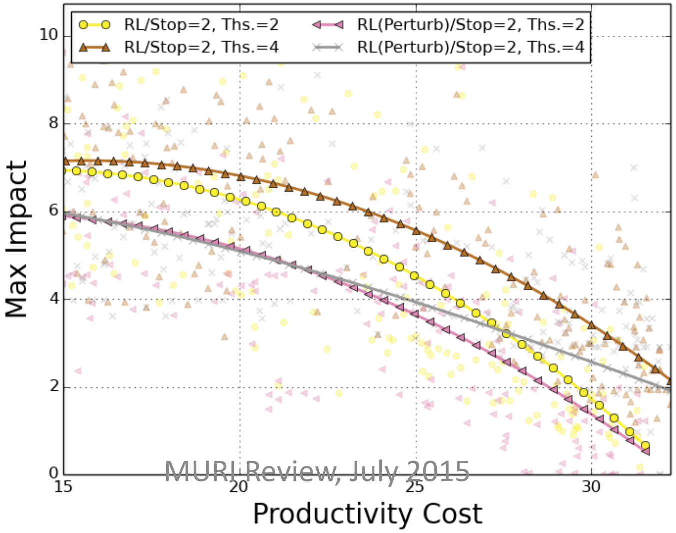
Perturbation

- Utility values are likely to be wrong !
- For greater resilience and robustness of the results, we perturbed the utility values and obtained similar results.

Experiment







In practice, perturbation of attacker strategy yields better defense (i.e. lower impact).

Contact Information

V.S. Subrahmanian
Dept. of Computer Science & UMIACS
University of Maryland
College Park, MD 20742.
Tel: 301-405-6724
Email: vs@cs.umd.edu
Web: www.cs.umd.edu/~vs/

Algorithm and Complexity Defender Strategy

- Decisional version is **NP**-*hard (Containment result difficult to state due to the expected values)*

- We use a NSGA2 genetic algorithm to solve the multi-objective optimization problem
  - Iterative algorithm that at each step evolves a population of individuals (solutions representing Pareto Points)
  - An individual is represented by a binary vector of size |AB|+|RB| (1 in the i-position means that the element is contained, 0 otherwise)
  - A population can be evolved by a random mutation of some individuals and by a cross-over operation among two individuals.
  - Non Dominating Sort Approach is usedto select the N individuals to propagate at the new evolution step
  - The last evolved population represents the approximated Pareto Frontier.