Are there always more vuls?

Jonathan Spring

VULCANO, October 20, 2021

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Document Markings

This material is based upon work funded and supported by the Department of Homeland Security under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

DM21-0934

What is the supply of undiscovered vulnerabilities?

This question is a useful input to vulnerability management policy and strategy.

- Vendors
- Consumers
- Public policy

I will bound the problem with a

- Conceptual analysis
- Economic analysis

What is the supply of undiscovered vulnerabilities?

This question is a useful input to vulnerability management policy and strategy.

- Vendors
- Consumers
- Public policy

I will bound the problem with a

- Conceptual analysis
- Economic analysis

What is the supply of undiscovered vulnerabilities?

This question is a useful input to vulnerability management policy and strategy.

- Vendors
- Consumers
- Public policy

I will bound the problem with a

- Conceptual analysis
- Economic analysis

What is the supply of undiscovered vulnerabilities?

This question is a useful input to vulnerability management policy and strategy.

- Vendors
- Consumers
- Public policy

I will bound the problem with a

- Conceptual analysis
- Economic analysis

What is the supply of undiscovered vulnerabilities?

This question is a useful input to vulnerability management policy and strategy.

- Vendors
- Consumers
- Public policy

I will bound the problem with a

- Conceptual analysis
- Economic analysis

Approaches so far

Dense vs. Scarce

- “I believe that vulns are scarce enough for [cornering the global vulnerability market] to work” (Geer, 2014)
- “But while vulnerabilities are plentiful, they’re not uniformly distributed. [P]ractices that eliminate many easy-to-find ones greatly improve software security.” (Schneier, 2014)

Let’s put this on a firmer footing

Approaches so far

Dense vs. Scarce

- "I believe that vulns are scarce enough for [cornering the global vulnerability market] to work" (Geer, 2014)
- "But while vulnerabilities are plentiful, they're not uniformly distributed. [P]ractices that eliminate many easy-to-find ones greatly improve software security." (Schneier, 2014)

Let's put this on a firmer footing

Approaches so far

Dense vs. Scarce

- “I believe that vulns are scarce enough for [cornering the global vulnerability market] to work” (Geer, 2014)
- “But while vulnerabilities are plentiful, they’re not uniformly distributed. [P]ractices that eliminate many easy-to-find ones greatly improve software security.” (Schneier, 2014)

Let’s put this on a firmer footing

Density (mathematical formalism)

A system that is a model of:

$$\forall x\ \forall y\ (x < y \rightarrow \exists z(x < z \wedge z < y))$$

That is: For any elements x and y, there exists some other element z such that z is between x and y.

A system that is dense has infinitely many elements.

- Systems can have infinite elements without being dense
- Dense systems can have countably infinite or uncountably infinite elements

“Dense” turns out not to be a good formal lens for this question

Density (mathematical formalism)

A system that is a model of:

$$\forall x\ \forall y\ (x < y \rightarrow \exists z(x < z \wedge z < y))$$

That is: For any elements x and y, there exists some other element z such that z is between x and y.

A system that is dense has infinitely many elements.

- Systems can have infinite elements without being dense
- Dense systems can have countably infinite or uncountably infinite elements

“Dense” turns out not to be a good formal lens for this question

Density (mathematical formalism)

A system that is a model of:

$$\forall x\ \forall y\ (x < y \rightarrow \exists z(x < z \wedge z < y))$$

That is: For any elements x and y, there exists some other element z such that z is between x and y.

A system that is dense has infinitely many elements.

- Systems can have infinite elements without being dense
- Dense systems can have countably infinite or uncountably infinite elements

“Dense” turns out not to be a good formal lens for this question

Density (mathematical formalism)

A system that is a model of:

$$\forall x\ \forall y\ (x < y \rightarrow \exists z(x < z \wedge z < y))$$

That is: For any elements x and y, there exists some other element z such that z is between x and y.

A system that is dense has infinitely many elements.

- Systems can have infinite elements without being dense
- Dense systems can have countably infinite or uncountably infinite elements

"Dense" turns out not to be a good formal lens for this question

Density (mathematical formalism)

A system that is a model of:

$$\forall x\ \forall y\ (x < y \rightarrow \exists z(x < z \wedge z < y))$$

That is: For any elements x and y, there exists some other element z such that z is between x and y.

A system that is dense has infinitely many elements.

- Systems can have infinite elements without being dense
- Dense systems can have countably infinite or uncountably infinite elements

“Dense” turns out not to be a good formal lens for this question

Different kinds of “a lot”

There are different categories of “always more” elements

Countably infinite

- Computers can entirely process sets of things that are this big
- Counting numbers 0, 1, 2, 3, etc.

Uncountably infinite

- Computers cannot entirely process sets of things that are this big
- The set of all possible sets of counting numbers

Different kinds of “a lot”

There are different categories of “always more” elements

Countably infinite

- Computers can entirely process sets of things that are this big
- Counting numbers 0, 1, 2, 3, etc.

Uncountably infinite

- Computers cannot entirely process sets of things that are this big
- The set of all possible sets of counting numbers

Different kinds of “a lot”

There are different categories of “always more” elements

Countably infinite

- Computers can entirely process sets of things that are this big
- Counting numbers 0, 1, 2, 3, etc.

Uncountably infinite

- Computers cannot entirely process sets of things that are this big
- The set of all possible sets of counting numbers

How many vulnerabilities in one static piece of software?

Turing Machine – formal model of computers and computation

Classic (Turing, 1936) problem statement is: can we determine ahead of time whether a computer program, given a specific input, will halt with a result or run forever?

- No, we cannot. This is the Halting Problem.
- This has since been shown equivalent to 2 other fundamental mathematical concepts (incompleteness, recursive computability)

To find all the vuls in one static piece of software, first you'd have to solve the Halting Problem.

- This implies software has uncountably infinitely many vuls

How many vulnerabilities in one static piece of software?

Turing Machine – formal model of computers and computation

Classic (Turing, 1936) problem statement is: can we determine ahead of time whether a computer program, given a specific input, will halt with a result or run forever?

- No, we cannot. This is the Halting Problem.
- This has since been shown equivalent to 2 other fundamental mathematical concepts (incompleteness, recursive computability)

To find all the vuls in one static piece of software, first you'd have to solve the Halting Problem.

- This implies software has uncountably infinitely many vuls

How many vulnerabilities in one static piece of software?

Turing Machine – formal model of computers and computation

Classic (Turing, 1936) problem statement is: can we determine ahead of time whether a computer program, given a specific input, will halt with a result or run forever?

- No, we cannot. This is the Halting Problem.
- This has since been shown equivalent to 2 other fundamental mathematical concepts (incompleteness, recursive computability)

To find all the vuls in one static piece of software, first you'd have to solve the Halting Problem.

- This implies software has uncountably infinitely many vuls

How many vulnerabilities in one static piece of software?

Turing Machine – formal model of computers and computation

Classic (Turing, 1936) problem statement is: can we determine ahead of time whether a computer program, given a specific input, will halt with a result or run forever?

- No, we cannot. This is the Halting Problem.
- This has since been shown equivalent to 2 other fundamental mathematical concepts (incompleteness, recursive computability)

To find all the vuls in one static piece of software, first you'd have to solve the Halting Problem.

- This implies software has uncountably infinitely many vuls

The importance of context for application

Vulnerabilities depend on context.

- "A vulnerability is a set of conditions or behaviors that allows the violation of an explicit or implicit security policy. Vulnerabilities can be caused by software defects, configuration or design decisions, unexpected interactions between systems, or environmental changes."

Even if we have a proof that a specific piece of software has no vuls, that is relative to a specific security policy.

- If the policy changes (for example, software is installed at a new organization), the proof may or may not hold.

The importance of context for application

Vulnerabilities depend on context.

- "A vulnerability is a set of conditions or behaviors that allows the violation of an explicit or implicit security policy. Vulnerabilities can be caused by software defects, configuration or design decisions, unexpected interactions between systems, or environmental changes."

Even if we have a proof that a specific piece of software has no vuls, that is relative to a specific security policy.

- If the policy changes (for example, software is installed at a new organization), the proof may or may not hold.

The importance of context for application

Vulnerabilities depend on context.

- "A vulnerability is a set of conditions or behaviors that allows the violation of an explicit or implicit security policy. Vulnerabilities can be caused by software defects, configuration or design decisions, unexpected interactions between systems, or environmental changes."

Even if we have a proof that a specific piece of software has no vuls, that is relative to a specific security policy.

- If the policy changes (for example, software is installed at a new organization), the proof may or may not hold.

The importance of context for application

Vulnerabilities depend on context.

- "A vulnerability is a set of conditions or behaviors that allows the violation of an explicit or implicit security policy. Vulnerabilities can be caused by software defects, configuration or design decisions, unexpected interactions between systems, or environmental changes."

Even if we have a proof that a specific piece of software has no vuls, that is relative to a specific security policy.

- If the policy changes (for example, software is installed at a new organization), the proof may or may not hold.

Is the Halting Problem applicable to modern software?

The CPU is a Turing Machine that runs Turing Machines (software)

Hardware is not (usually) designed with physical separation between resources available to programs and is finite. However:

- The internet serves as an effectively infinite tape
- Even offline, 3.5 GB of RAM has $256^{3,500,000,000}$ possible states
  - It's not practical to exhaustively check if each state is vulnerable
  - Fuzzing is useful, but this is why fuzzing is not just exhaustive

In practice, program verification selects a specific, narrow security policy element to verify (Pym, Spring, O'Hearn, 2018)

- Because Halting Problem.

Is the Halting Problem applicable to modern software?

The CPU is a Turing Machine that runs Turing Machines (software)

Hardware is not (usually) designed with physical separation between resources available to programs and is finite. However:

- The internet serves as an effectively infinite tape
- Even offline, 3.5 GB of RAM has $256^{3,500,000,000}$ possible states
  - It's not practical to exhaustively check if each state is vulnerable
  - Fuzzing is useful, but this is why fuzzing is not just exhaustive

In practice, program verification selects a specific, narrow security policy element to verify (Pym, Spring, O'Hearn, 2018)

- Because Halting Problem.

Is the Halting Problem applicable to modern software?

The CPU is a Turing Machine that runs Turing Machines (software)

Hardware is not (usually) designed with physical separation between resources available to programs and is finite. However:

- The internet serves as an effectively infinite tape
- Even offline, 3.5 GB of RAM has $256^{3,500,000,000}$ possible states
  - It's not practical to exhaustively check if each state is vulnerable
  - Fuzzing is useful, but this is why fuzzing is not just exhaustive

In practice, program verification selects a specific, narrow security policy element to verify (Pym, Spring, O'Hearn, 2018)

- Because Halting Problem.

Is the Halting Problem applicable to modern software?

The CPU is a Turing Machine that runs Turing Machines (software)

Hardware is not (usually) designed with physical separation between resources available to programs and is finite. However:

- The internet serves as an effectively infinite tape
- Even offline, 3.5 GB of RAM has $256^{3,500,000,000}$ possible states
  - It's not practical to exhaustively check if each state is vulnerable
  - Fuzzing is useful, but this is why fuzzing is not just exhaustive

In practice, program verification selects a specific, narrow security policy element to verify (Pym, Spring, O'Hearn, 2018)

- Because Halting Problem.

Relative effort

Can the effort and cost to an attacker to find another vulnerable state be higher than the value of the knowledge of the vulnerable state?

Developers get ahead of attackers with pre-release vulnerability discovery

Anyone (attackers, customers, developers) can do post-release discovery

If a developer does not use open-source discovery tools, someone else will post-release and find vuls quickly

- Best case, the developer has a bug bounty, responsive development, and automatic update delivery
- Attackers can also analyze patches and attack unpatched systems

Relative effort

Can the effort and cost to an attacker to find another vulnerable state be higher than the value of the knowledge of the vulnerable state?

Developers get ahead of attackers with pre-release vulnerability discovery

Anyone (attackers, customers, developers) can do post-release discovery

If a developer does not use open-source discovery tools, someone else will post-release and find vuls quickly

- Best case, the developer has a bug bounty, responsive development, and automatic update delivery
- Attackers can also analyze patches and attack unpatched systems

Relative effort

Can the effort and cost to an attacker to find another vulnerable state be higher than the value of the knowledge of the vulnerable state?

Developers get ahead of attackers with pre-release vulnerability discovery

Anyone (attackers, customers, developers) can do post-release discovery

If a developer does not use open-source discovery tools, someone else will post-release and find vuls quickly

- Best case, the developer has a bug bounty, responsive development, and automatic update delivery
- Attackers can also analyze patches and attack unpatched systems

Relative effort

Can the effort and cost to an attacker to find another vulnerable state be higher than the value of the knowledge of the vulnerable state?

Developers get ahead of attackers with pre-release vulnerability discovery

Anyone (attackers, customers, developers) can do post-release discovery

If a developer does not use open-source discovery tools, someone else will post-release and find vuls quickly

- Best case, the developer has a bug bounty, responsive development, and automatic update delivery
- Attackers can also analyze patches and attack unpatched systems

Attacker economic incentives

If the economic return on compute time for vulnerability discovery is lower than that of mining bitcoin, then attackers should stop spending their limited compute resources on vulnerability discovery.

- Return is dynamic
- As fewer vuls are found, if demand is constant their price goes up
- Unless demand goes down, eventually the price will rise high enough
- Punishment may increase the attacker's risk, also increasing price

Also, the supply of vuls to be found may be inelastic

- There are uncountably infinitely many

Attacker economic incentives

If the economic return on compute time for vulnerability discovery is lower than that of mining bitcoin, then attackers should stop spending their limited compute resources on vulnerability discovery.

- Return is dynamic
- As fewer vuls are found, if demand is constant their price goes up
- Unless demand goes down, eventually the price will rise high enough
- Punishment may increase the attacker's risk, also increasing price

Also, the supply of vuls to be found may be inelastic

- There are uncountably infinitely many

Attacker economic incentives

If the economic return on compute time for vulnerability discovery is lower than that of mining bitcoin, then attackers should stop spending their limited compute resources on vulnerability discovery.

- Return is dynamic
- As fewer vuls are found, if demand is constant their price goes up
- Unless demand goes down, eventually the price will rise high enough
- Punishment may increase the attacker's risk, also increasing price

Also, the supply of vuls to be found may be inelastic

- There are uncountably infinitely many

System owner economic costs

Aspects to cost in vulnerability management given a known vulnerability:

- Vulnerability risk (highly variable)
- Change risk (highly variable)
- Labor costs (predictable)
- Operational downtime costs (sector-specific concerns)

System owner economic costs

Aspects to cost in vulnerability management given a known vulnerability:

- Vulnerability risk (highly variable)
- Change risk (highly variable)
- Labor costs (predictable)
- Operational downtime costs (sector-specific concerns)

Interplay between attacker and defender incentives

Developer will not usually do all fuzzing and verification (Anderson, 2001)

- There will always be more vuls to find

System Owner will not immediately apply all patches

- due to costs and change risk

Some attacker will always gain value in finding vuls

- Punishments can increase the price but not reduce supply

This leaves demand for vulnerabilities as an interesting topic

- We can discuss, but I'm not sure how this can be regulated or reduced

Interplay between attacker and defender incentives

Developer will not usually do all fuzzing and verification (Anderson, 2001)

- There will always be more vuls to find

System Owner will not immediately apply all patches

- due to costs and change risk

Some attacker will always gain value in finding vuls

- Punishments can increase the price but not reduce supply

This leaves demand for vulnerabilities as an interesting topic

- We can discuss, but I'm not sure how this can be regulated or reduced

Interplay between attacker and defender incentives

Developer will not usually do all fuzzing and verification (Anderson, 2001)

- There will always be more vuls to find

System Owner will not immediately apply all patches

- due to costs and change risk

Some attacker will always gain value in finding vuls

- Punishments can increase the price but not reduce supply

This leaves demand for vulnerabilities as an interesting topic

- We can discuss, but I'm not sure how this can be regulated or reduced

Interplay between attacker and defender incentives

Developer will not usually do all fuzzing and verification (Anderson, 2001)

- There will always be more vuls to find

System Owner will not immediately apply all patches

- due to costs and change risk

Some attacker will always gain value in finding vuls

- Punishments can increase the price but not reduce supply

This leaves demand for vulnerabilities as an interesting topic

- We can discuss, but I'm not sure how this can be regulated or reduced

Thank you for your time

Questions?

jspring AT cert . org