State of Fuzzing 2017

SYNOPSYS®

TABLE OF CONTENTS

ABOUT THE DATA

This report includes data from more than 4.8 billon individual fuzz tests run in 2016. Where possible, data from 2016 has been compared with 2015. There were 6.1 billion fuzz tests conducted in 2015. Such variations in test runs are subject to items in the news, changes in industry verticals, and the presence of new tools using Synopsys Fuzz Testing.

The data in this report has been anonymized and reveals only the number of tests, time to first failure, and length of each test. Tests by vertical have been inferred from protocols typically associated with those verticals and do not reflect results derived from actual members of those verticals. To learn how Synopsys categorizes over 250 protocols by technology, please see:

https://synopsys.com/software-integrity/security-testing/fuzz-testing/defensics.html

Executive summary

Fuzzing is a proven technology used to find vulnerabilities in software by sending malformed input to a target and observing the result. If the target behaves unexpectedly or crashes, then further investigation is required. That investigation may expose a vulnerability that may be exploited for malicious purposes. Fuzzing is equally valuable to those who develop software and those who consume it. It plays a role in the implementation, verification, and release phases of the software development life cycle (SDLC) and can be a vital indicator of undetected vulnerabilities (zero days) that may affect the integrity of systems already in use. The real goal of fuzzing is not merely to crash a program but to hijack it.

The State of Fuzzing 2017 is based on 4.8 billion test results from anonymous customers using Synopsys Fuzz Testing (Defensics) from Jan. 1 through Dec. 31, 2016. The report provides insight into the protocols used by different industry verticals and the relative maturity of those protocols. The relative maturity of protocols is measured in terms of how long it takes before there is a first software failure and the overall number of failures recorded.

- Mature protocol—These are either simple protocols or ones that are well-tested over time. By simple we mean that the protocol, as defined in its RFC, has relatively few features and that these features have been tested or implemented correctly over the years. An example of a well-tested protocol is ARP, an old-school protocol that maps hardware media access control (MAC) addresses to IP addresses.

- Immature protocol—These are either complex protocols or ones that are not well-tested. Complex means that the protocol, as defined in its RFC, has many features, some of which may not have been tested or may not have been implemented correctly over the years. An example of a complex protocol is SIP UAS, used in computer telephony or VoIP-type technologies.

When grouped by industry vertical, the protocols demonstrating the least maturity (i.e., highest number of failures recorded over the shortest amount of time) and therefore the highest risk were found in Industrial Control Systems (ICS). This vertical favors its own niche protocols, which may not have been well tested over the years. With the need for more systems to intercommunicate and the advent of the Industrial Internet of Things (IIoT), these niche protocols are now finding increased use. Average test runtimes for ICS were low (only around an hour or two), which suggests that longer fuzz test runtimes will reveal more crashes and perhaps more vulnerabilities.

Many of the ICS protocols are also present in the consumer Internet of Things (IoT). Although far more diverse in terms of overall protocol categories represented, IoT also contains many older protocols that are gaining new popularity now that software is everywhere and must intercommunicate. Vulnerabilities identified here have profound consequences. For example, handheld devices and home sensors may become compromised if off-the-shelf protocols were used but not implemented or tested thoroughly. As IoT expands, it is important that vendors start testing for unknowns, if they haven't already. This is because many of the firmware-based personal IoT systems may be hard or impossible to update post-release.

Occupying the mid-level risk for new vulnerabilities are the healthcare and automotive industries. These industries use a healthier mix of niche protocols, such as DICOM and CAN Bus, as well as mature Core IP protocols, such as TCP and ARP. These industries may also use Wireless (Bluetooth) and Media files (JPEG and MP3). While less risky because their access is more limited than ICS and IoT, Healthcare and Automotive still provide fertile ground for new vulnerability research.

Message Queue Telemetry Transport, or MQTT

MQTT is a lightweight messaging protocol designed for solutions where internet connectivity is limited or costly. Not surprisingly, this protocol is used in infrastructure such as power plants and manufacturing. However, independent researchers have also found that it's being used for airplane coordinates, prison door commands, low-level automotive functions, electrical meters, medical equipment, mobile phones, and home alarm and home automation systems. These same researchers have also found critical vulnerabilities in these systems associated with their use of MQTT and related protocols. There were 85 failures against 831,483 test runs in 2016, and the time to first failure was 32 minutes. The overall test time, however, was only 1.6 hours on average. This protocol would benefit from longer test runs.

Finally, there are some industry verticals, such as Government and Financial Services, that are relatively less risky because of the diversity and number of protocol categories represented. The protocols used within those categories are also more mature. Core IP and Media protocols are some of the more mature protocols observed in 2016. They had fewer overall failures and longer test times to that first failure. Again, this is not to say these protocols have a clean bill of health, only that more persistent fuzzing (i.e., longer overall fuzz testing runtime) is needed to find fewer but just as damaging vulnerabilities.

These individual industry metrics are valuable to organizations because they help organizations direct software testing toward those protocols with a higher likelihood of failure and shorter time to failure (i.e., where the zero days are). A high number of crashes combined with a short amount of time to first failure (TTFF, or the time it takes until the first crash is recorded) should indicate a higher likelihood of exploitable vulnerabilities. Similarly, the absence of crashes over a longer period should suggest a low likelihood of exploitable vulnerabilities. While the number of software failures is not one-to-one indicative of the number of exploitable vulnerabilities likely to be found (e.g., one vulnerability may result in numerous failures), the number is nonetheless representative of areas where future exploits might be possible. Therefore, these scenarios require further investigation.

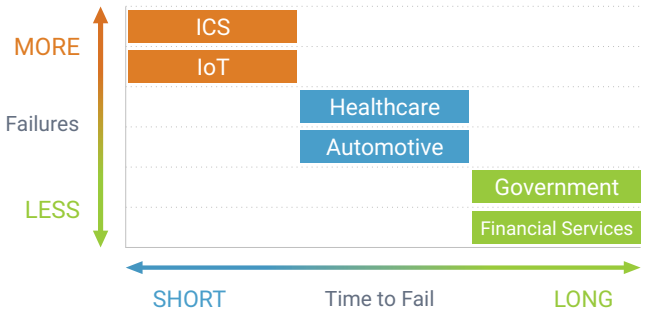

Figure 1: Industry verticals by relative risk

Heartbleed

Sometimes the implementation of a protocol is flawed. The classic example of why you should fuzz is Heartbleed.

Within the RFC for SSL is a feature that continually pings between the client and the server—a heartbeat—to prove the connection still exists. Within some earlier versions of OpenSSL, however, the implementation was such that instead of limiting the call and response to the standard 64 bits, the field was unbounded, allowing for data leakage. Researchers at Synopsys (at the time Codenomicon) and Google co-discovered the Heartbleed vulnerability, technically know as CVE 2014-1060.

From our State of Software Security 2017 report, we see some applications still include vulnerable versions of OpenSSL (1.0.1 to 1.0.1f) despite the presence of a patch.

2016 snapshot

In 2016 we observed the following:

» Customers ran more than 4.8 billion individual fuzz tests using Synopsys Fuzz Testing.

» Customers tested over 250 different protocol-based suites.

» Just over half the protocol suites composing our overall top 20 are new to the list. These include the following:

- STP
- TCP for IPv4
- ICMPv4
- TLS 1.2
- OPC UA
- SCTP
- TCP for IPv6
- LLDP
- SNMPv2c
- SIP UAS
- E-LMI

» The overall average time to first failure (TTFF) in 2016 was 1.4 hours.

This overall average represents a decrease from 1.7 hours in 2015 (higher number, fewer faults).

- **The most mature protocol** tested in 2016 was TLS Client (Core IP).
  Average TTFF (average time for first failure) for TLS Client was 9 hours in 2016.
- **The least mature protocol** tested in 2016 was IEC-61850 MMS (ICS).
  Average TTFF for IEC-61850 MMS was 6.6 seconds.

» The overall number of test failure numbers can also suggest protocol maturity.

- A more mature protocol is ARP (zero failures against 344,443 tests).
- A less mature protocol is SIP UAS (320,509 failures against 107,431,568 tests).

» The following protocol test suites by industry were added in 2016:

Internet of Things (IoT) (7)

- OPC UA (ICS)
- SSHv2 Server (Remote Management)
- CoAP Server (ICS)
- Traffic with SDK (General)
- SSH Server (Core IP)
- TCP for IPv4 Client (Core IP)
- JPEG (Media)

Industrial Control Systems (ICS) (0)

Healthcare (5)

- TCP for IPv4 Client (Core IP)
- AVRCP (Wireless)
- Bluetooth LE Advertising Data (Wireless)
- HFP (Wireless)
- SMP (Wireless)

Automotive (2)

- JPEG (Media)
- MPEG4 (Media)

Financial Services (6)

- HTTP Server (Core IP)
- CMPv6 (Core IP)
- TCP for IPv6 Server (Core IP)
- SNMPv2c (Remote Management)
- DHCPv4 Server (Core IP)
- CIFS Server (Storage)

Government (3)

- TCP for IPv6 Server (Core IP)
- E-LMI (Metro Ethernet)
- DHCPv4 Server (Core IP)

Finding zero days in 2017

In a modern network environment, well- established organizations like financial services providers, governments, and large corporations protect their critical assets by placing them behind a well-patched multilayer defense system. That doesn't mean they aren't impenetrable, only that all or most of the known vulnerabilities have been mitigated. That's why elite criminal hackers spend their time looking for unknown vulnerabilities, or zero days. Exploiting zero days can allow criminal activity to carry on for years with unmonitored access to sensitive information.

So how do you find unknown vulnerabilities in software? One way to find this type of vulnerability in a black box scenario is to throw different inputs at the target and then observe the result. Think of this as intentionally sending noise or gibberish to the software and observing the result.

Although not at all the same, a SQL injection attack is similar in that the presence of a special character in user input may cause the program to behave differently than expected or crash. With fuzzing, the result is more often a system crash, a denial of service, or an anomaly (e.g., too much data returned). This creates an opportunity for a bad actor to perform any number of unsanctioned activities.

Fuzzing is the structured process of sending incorrect or unexpected inputs to software and checking for failure in a controlled environment. It began as an exercise in creating different ways to create malformed input or negative input (see sidebar). Malformed input doesn't have to be limited to network packets. Fuzzing can also be applied to APIs, registry keys, or files—any type of data that might cross a boundary between client and server and allow somebody with no privileges to suddenly have privileges on that system entering the

A brief history of fuzzing

"It was a dark and stormy night."

This is how Barton Miller began his introduction to "Fuzzing for Software Security Testing and Quality Assurance" (2008), one of the first books to describe the process of fuzzing. Miller accidently discovered the process of fuzzing during a Midwestern thunderstorm in the fall of 1988.

While he was logged in to the Unix system at the University of Wisconsin over a 1200-baud modem, the noise on the line from the storm kept interfering with Miller's ability to type commands. What surprised him was how the random noise would occasionally cause the common Unix programs to crash.

Miller began to wonder whether there ought to be a formal method for generating noise—a process he decided should be called "fuzz"—to see why and where the programs failed. Thus began the science of fuzzing, the use of malformed input to generate crashes in software programs.

target in any way. Fuzz testing should be performed across the entire attack surface of the target and as part of a larger defense strategy.

This is important today as software is now everywhere and the real world is a messy place. It is full of unexpected conditions and badly formed input from poorly written applications. Software must be able to deal with people who will supply poorly formed input, perform actions in an unexpected order, and generally misuse the software.

Where to look for vulnerabilities?

The fuzzing process starts by generating the data as it should be—a string, a username, a password, whatever it is. With fuzzing, new input that is automatically generated may or may not be accepted by the application. If the application crashes, the developer can go back to determine why the application crashed.

Fuzzing is an infinite space problem. For any piece of software, the set of invalid inputs will always be unbounded. Therefore, any effective fuzz test must include all, if not most, use cases that are likely to trigger bugs in the target software. Whether a given test suite finds any failures varies based on the robustness of the individual target, or system under test (SUT). Based on this knowledge, it is important to understand that this data cannot always be considered comparable from system to system.

How do you fuzz?

Fuzzing has grown from a low-budget technique used by individual hackers to large-scale distributed testing on platforms like Azure and Amazon, which are capable of spinning up a large number of virtualized computers that fuzz test a program in parallel. Hackers will scour their fuzz inputs that led to crashes to see what error conditions were triggered. Knowing that criminal hackers will do this if they don't do it themselves, large software vendors create policies for the number of

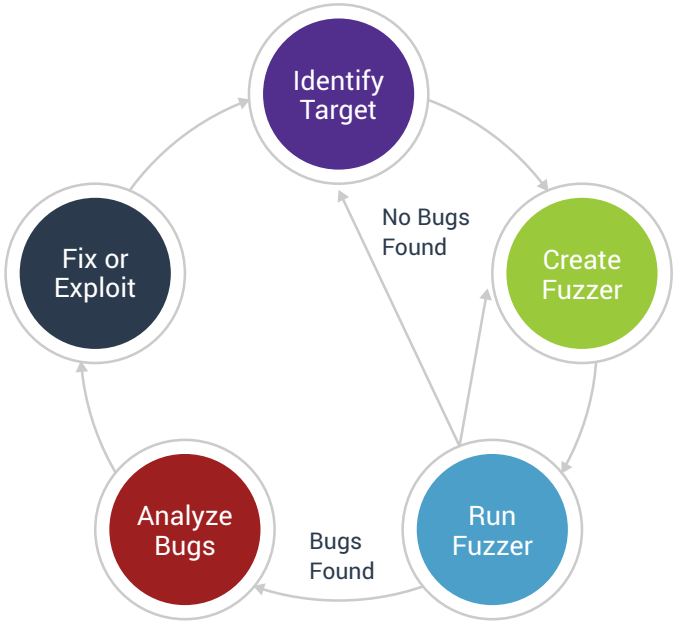

Figure 2: Typical bill of process

Fuzz Testing Maturity Model

The Fuzz Testing Maturity Model is a vendor-agnostic guide to help organizations standardize their fuzzing practice. It specifies six levels of maturity from Level 0 (immature) to Level 5 (optimized), defining requirements for each.

The maturity model defines industry fuzzing terms so that organizations can talk to other organizations. It provides a framework to assess the maturity of your own software development as well as any supply chain partners used.

Synopsys.com/fuzz-test

tests run and the number of allowable failures before a piece of software will ship to customers. This speaks not only to the overall security of the product, but also to the quality of the software code itself (i.e., there will be fewer crashes overall). Errors that cause crashes may be "fun," but the real money is in events that lead to system compromise or escalation of privilege.

There are free open source fuzzing tools, and they are the types of tools elite hackers use, but they come with their own challenges. They require the end user to develop working scripts to create malformed input, deliver it to the target, and gather the output. There are pros and cons to these tools. In some cases, the fuzzing message generation scripts themselves are flawed, the model is incorrect, or the delivery method is weak.

A very good example of a targeted open source fuzz testing service is the Google Open Source Fuzzing (OSS-Fuzz) platform. Google has partnered with 47 open source software organizations to provide continuous testing of their code, performing up to 4 trillion tests a week. The data in the State of Fuzzing report is not applicable only to open source or Web application targets; rather it highlights the importance of fuzzing all types of software. The data here comes from firmware and other network surfaces, open source and proprietary. It represents real-world customers across many industry verticals, from Automotive to Government.

What does fuzzing target?

When talking about fuzzing, it is important to establish the target. In this case, it is a single piece of software or a collection of software also referred to as the system under test (SUT), device under test (DUT), interface, firmware, system, service, and so on. Fuzzing can also be black box or gray box testing, meaning it requires little or no knowledge of the inner workings of the target. This flexibility makes fuzzing an extremely useful tool for software, regardless of the availability of source code or detailed internal information. As a black box technique, fuzzing is useful to anyone who wants to understand the real-life robustness and reliability of the systems they are operating or planning to deploy. It is also the reason why fuzzing is the number one technique used by black hat operatives and hackers to find software vulnerabilities.

Definitions every fuzz tester must know

What is a protocol?

When an application must communicate with another application or server, it uses a protocol. A Request for Comments (RFC) is a formal document from the Internet Engineering Task Force (IETF) defining the format of data sent over the internet or another network. Within each protocol are various features that may be enabled or disabled depending on use. How the protocol is implemented may vary.

What is a file format?

A file format is a specific requirement that a file must follow—for example, a JPEG file.

What is an API?

An application programming interface (API) includes subroutine definitions, protocols, and the necessary tools for building application software.

Why test a target system?

Even without access to source code, the ability to more closely monitor the vitals of the target software improves the quality of the testing. Log files, process information, and resource usage provide valuable information that can be used during fuzzing to understand how anomalous input affects the target system.

Other members of the software development ecosystem can benefit from fuzzing. For example, regulatory agencies wishing to improve software quality in an industry can require a minimum level of fuzz testing for products from suppliers. Such software supply chain requirements benefit all involved, including customers, who get better products and services. If the source code is available, debugger tooling and other instrumentation can be used to make fuzzing more of a gray box technique. This improves the accuracy of the tests and allows more rapid resolution of located defects.

Fuzzing is the number one technique used by black hat operatives and hackers to find software vulnerabilities.

"Properly executed fuzzing techniques can provide a low-cost, efficient means of finding vulnerabilities, covering more code paths and value iterations than a manual analysis can perform in a short period of time."

--Bow Sineath, director of technology at Alpha Defense

How does fuzz testing work?

The world of fuzzing can be divided into network protocol fuzzing, file format fuzzing, and other disciplines. Different disciplines have slight variations in execution, injection modes, and failure modes. The more information that is known or available from the target, the more it aids in remediation of the issues discovered during testing.

There are three primary methods of network protocol fuzzing:

1. **Random:** The simplest and least effective fuzzing method is random fuzzing. Random fuzzing is usually ineffective because the test cases are nothing like valid input. The target examines and quickly rejects the test cases. For the most part, the test cases fail to penetrate the target code, but random fuzzing can be somewhat effective for new or immature code.

2. **Template:** An intermediate testing method is called template fuzzing, also known as block or mutational fuzzing. It provides slight alterations, iterating (perhaps by as little as one byte) until the user is satisfied the testing has found enough errors. Changing a known good input incrementally can help identify potential vulnerabilities. However, this process can take a long time. It can also miss vulnerabilities owing to its narrow focus (i.e., starting with a known good input) and become difficult to manage as testing evolves.

3. **Generational:** A more advanced method is generational or model-based fuzzing. This method simulates randomness but is designed around the protocol RFC data model, giving it a structured framework. In other words, generational fuzzing creates malformed input based on the original parameters of the protocol, API, or defined structure. The advantage of this method is controlled randomness, which reduces time wasted focusing on what works, thereby increasing efficiency. It allows the consideration of the broader scheme (what could work) without shooting the moon. Intelligent fuzzing is the method that helped Synopsys researchers co-discover the Heartbleed vulnerability in 2014 (see sidebar, page 7) and many other vulnerabilities.

The state of fuzzing 2017

There was a lot of movement within the top 20 protocols tested last year. In fact, just over half the protocols in our 2017 top 20 are new to the list. Among the protocol suites listed for the first time are STP, TCP for IPv4 Server, ICMPv4, TLS 1.2, OPC UA, SCTP, TCP for IPv6, LLDP, SNMPv2c, SIP UAS, and E-LMI. These represent diverse categories of protocols such as VPN, ICS, Cellular Core, Core IP, Storage, Metro Ethernet, Remote Management, and VoIP. The increase in new protocols reflects an overall greater diversity of industries that took an interest in fuzzing over the last year.

2016 Rank	Protocol suite	Type	Change	2015 Rank
1	STP	Link Management	new	new
2	HTTP Server	Core IP	0	2
3	TCP for IPv4 Server	Core IP	new	new
4	Traffic Capture Fuzzer	General Purpose	-3	1
5	IPv4	Core IP	-2	3
6	ARP Server	Core IP	-2	4
7	DNS Server	Core IP	-2	5
8	ICMPv6	Core IP	1	9
9	ICMPv4	Core IP	new	new
10	IPv6	Core IP	-2	8
11	TLS Server	VPN	-1	10
12	TLS 1.2 Server	VPN	new	new
13	OPC UA Server	ICS	new	new
14	SCTP Server	Cellular Core	new	new
15	TCP for IPv6 Server	Core IP	new	new
16	SMB2 Server	Storage	1	17
17	LLDP	Metro Ethernet	new	new
18	SNMPv2c	Remote Management	new	new
19	SIP UAS	VoIP	new	new
20	E-LMI	Metro Ethernet	new	new

Table 1: Top 20 protocol tests run in 2016

Top 20 protocol tests run in 2016

From the overall top 20, some of the more interesting individual entries include these:

- ARP Client/Server test. The Address Resolution Protocol (ARP) is used with IPv4 network addresses to hardware addresses (e.g., it is how your laptop connects to the printer across the room). In 2016, there were zero failures recorded against 272,044,482 test runs.

- IGMP. The Internet Group Management Protocol (IGMP) is a communications protocol. It is used by hosts and adjacent routers on IPv4 networks to establish multicast group memberships. There were 765 failures recorded against 22,846,689 test runs.

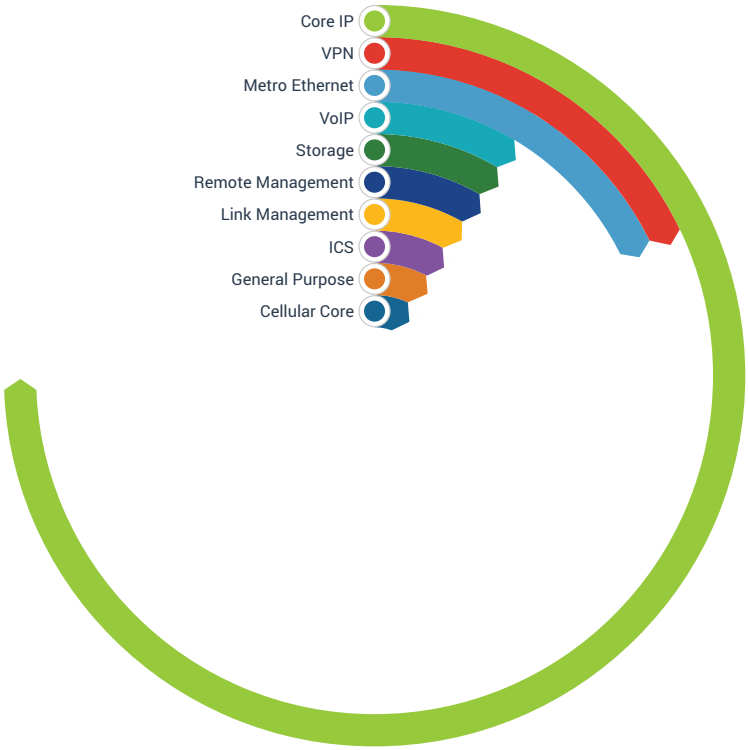

Figure 3: Distribution of top 20 protocol test categories

- ICMPv4. The Internet Control Message Protocol (ICMP) is an error-reporting protocol for network devices. For example, network routers use ICMP to generate error messages to the source IP address when problems prevent delivery of IP packets. There were 45,121 failures recorded against 164,780,794 test runs. Even though the number of failures is less than 1% of the total number of test runs, testing in this example shows how fuzzing can find that needle in the haystack. Remember, bad actors need only be right once.

> Remember, bad actors need only be right once.

The top 20 represents a wide range of protocol maturity. The average runtime for fuzzing ARP was 30 hours, and after 272 million tests, no failures were recorded. On the other hand, the average runtime for fuzzing ICMPv4 was 2.3 days, and after 164 million tests, over 45,000 failures were recorded.

Protocol suite	Time to first failure	Number of failures	Number of tests	Test runtime (avg)
ARP Client (Core IP)	0	0	272,044,482	30 hours
IGMP (Core IP)	46 seconds	765	22,846,689	37 minutes
ICMPv4 (Core IP)	2.7 hours	45,121	164,780,794	2.3 days

Top 20 protocols by industry

It is also valuable to abstract protocol tests based on use by specific industry verticals. Although the source of these tests is anonymous, it is nonetheless possible to infer use based on known standards and regulations in key verticals. Additionally, it is interesting to see the change in use of specific protocols from 2015 to 2016. Specific verticals covered in this section include Industrial Control Systems (ICS), Internet of Things (IoT), Healthcare, Automotive, Financial Services, and Government.

Industrial Control Systems (ICS)

Utilizing only three protocol categories, Industrial Control Systems (ICS) represents the least diverse range of the industry verticals presented in the State of Fuzzing 2017. The narrowness of breadth and the high number of niche protocols represented lead to a higher risk within ICS in general as compared to the other verticals. For example, four of the five least mature protocols called out by average time to first failure (TTFF) are ICS specific, and TTFFs range from a few seconds to a few minutes. In fuzzing, a longer time to failure is a good sign.

2016 Rank	Protocol suite	Type	Change	2015 Rank
1	TCP for IPv4 Server	Core IP	4	5
2	Traffic Capture Fuzzer	General Purpose	-1	1
3	IPv4	Core IP	-1	2
4	ARP Server	Core IP	-1	3
5	ICMPv4	Core IP	-1	4
6	OPC UA Server	ICS	12	18
7	CIP/EtherNet/IP Server	ICS	4	11
8	Profinet DCP Server	ICS	7	15
9	CoAP Server	ICS	5	14
10	DNP3 Client	ICS	10	20
11	Traffic Capture Fuzzer with SDK	General Purpose	-2	9
12	IEC-104 Client	ICS	4	16
13	TCP for IPv4 Client	Core IP	-8	5
14	IGMP	Core IP	-8	6
15	Universal Fuzzer	General Purpose	-7	8
16	MQTT Client	ICS	-3	13
17	IEC-61850-MMS Client	ICS	-5	12
18	ARP Client	Core IP	-8	10
19	MODBUS Master	ICS	-2	17
20	E-LMI	Metro Ethernet	new	new

Table 2: Top 20 protocols for Industrial Control Systems (ICS)

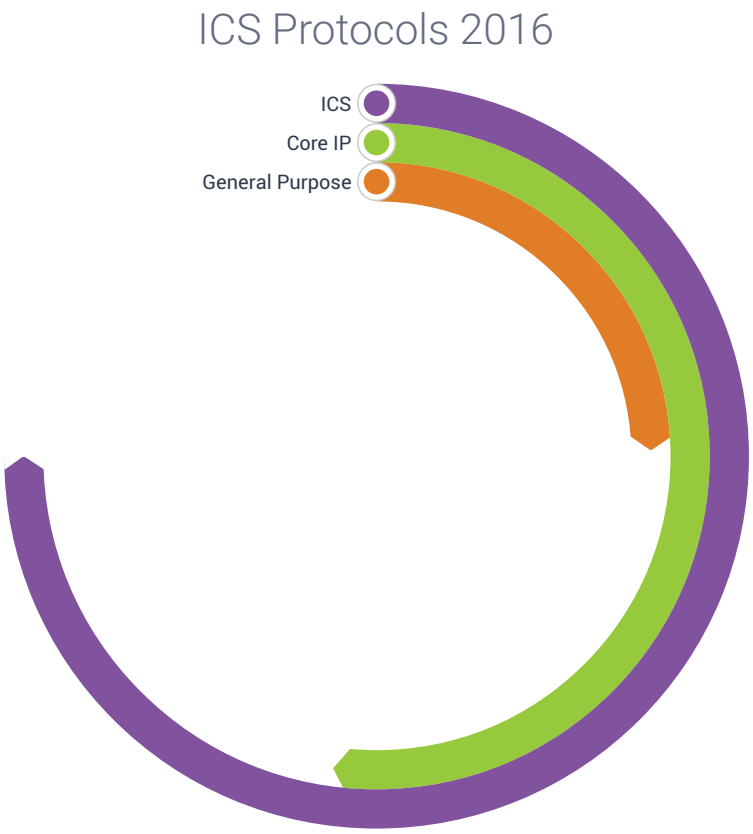

Figure 4: Distribution of protocol categories for Industrial Control Systems (ICS)

Some protocol test suites specific to ICS include these:

- IEC-104. This protocol defines systems used for telecontrol (supervisory control and data acquisition). It is used in electrical engineering and power system automation applications. There were 237 failures recorded in 2016 against 181,802,805 test runs; however, the time to first failure was a mere 6.6 seconds.
- IEC-61850. This is a protocol standard used for hydroelectric power plants' intelligent electronic devices. It allows electrical substation automation systems to be able to intercommunicate. There were four failures recorded in 2016 against 10,064,487 test runs, and the time to first failure was 1.2 minutes.
- ModBUS. This is a proprietary serial communications protocol originally published by Modicon (now Schneider Electric). It is used with programmable logic controllers (PLCs). There were 34 failures recorded against 79,173 test runs last year, and time to first failure was 1.8 minutes.

Four protocols—IEC-61850 MMS, IEC-104 Server, OPC UA, and MODBUS PLC—each had time to first failures recorded in under 5 minutes. However, a look at the relatively low overall number of failures attributed to each suggest these tests were not run long enough. By comparison the most mature protocol observed, ARP, had zero failures after 344,443 tests with an average runtime of 30 hours.

Protocol suite	Time to first failure	Number of failures	Number of tests	Test runtime (avg)
IEC-61850 MMS (ICS)	6.6 seconds	4	1,064,487	5.7 hours
IEC-104 Server (ICS)	1.2 minutes	237	12,136	6.5 minutes
MODBUS PLC (ICS)	1.8 minutes	37	1,533,725	16 minutes
OPC UA (ICS)	4.8 minutes	16,692	76,888,623	4.5 hours
MQTT (ICS)	13 minutes	85	831,483	1.6 hours

This uncertainty (time to failure vs. number of failures) alone should drive additional investigation into the protocols used within ICS in the coming years.

Internet of Things (IoT)

The Internet of Things (IoT) although still new, has a lot of overlap in protocols used with Industrial Control Systems. Despite spanning eight categories of protocol test suites, IoT is not the most diverse vertical; however, it comes close by mixing older, better-tested Core IP protocols with newer niche categories like Wireless and ICS.

2016 Rank	Protocol suite	Type	Change	2015 Rank
1	HTTP Server	Core IP	1	2
2	TCP for IPv4 Server	Core IP	7	9
3	Traffic Capture Fuzzer	General Purpose	-2	1
4	IPv4	Core IP	-2	2
5	ARP Server	Core IP	-2	3
6	ICMPv4	Core IP	1	7
7	TLS Server	VPN	-2	5
8	TLS 1.2 Server	VPN	2	10
9	OPC UA Server	ICS	new	new
10	TLS Client	VPN	-4	6
11	SSHv2 Server	Remote Management	new	new
12	CIP/EtherNet/IP Server	ICS	1	13
13	CoAP Server	ICS	new	new
14	Traffic Capture Fuzzer with SDK	General Purpose	new	new
15	XML-SOAP Server	Application	3	18
16	SSH Server	Remote Management	new	new
17	HTTP2 Server	Core IP	-3	14
18	TCP for IPv4 Client	Core IP	new	new
19	JPEG	Media	new	new
20	802.11 AP	Wireless	-3	17

Table 3: Top 20 protocols for Internet of Things (IoT)

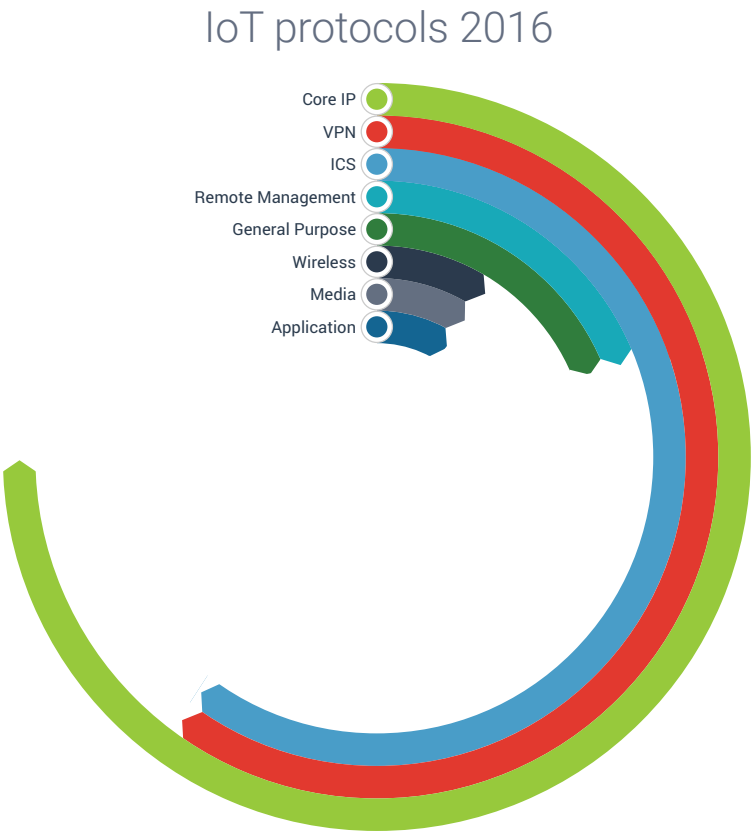

Figure 5: Distribution of protocol categories for Internet of Things (IoT)

Some protocol suites specific to IoT include these:

- **CoAP Server.** Constrained Application Protocol (CoAP) is an IoT-specific protocol used for a specialized Web transfer within constrained nodes and constrained networks. In other words it functions as a lightweight HTTP. In 2016, there were 6,275 failures recorded against 16,122,616 test runs with an average runtime of 3 hours.

- **CIP.** Common Industrial Protocol (CIP) is a unified communication architecture used throughout the manufacturing enterprise. There were 610 failures recorded in 2016 against 25,991,681 test runs with an average runtime of 5.3 hours.

- **OPC UA.** OPC Unified Architecture (OPC UA) is a machine-to-machine communication protocol. It is developed by a consortium of open connectivity of industrial automation devices and systems, the OPC Foundation. There were 16,692 failures recorded last year against 76,888,623 test runs with an average runtime of 4.5 hours.

For these IoT protocols, longer fuzzing runtime may not help the fact that these are immature by nature. Until time to first failure rises and the number of failures drops, additional testing times will only create more investigation time.

Protocol suite	Time to first failure	Number of failures	Number of tests	Test runtime (avg)
CoAP Server (ICS)	8.5 seconds	6,275	16,122,616	3 hours
CIP (ICS)	2 minutes	610	25,991,681	5.3 hours
OPC UA (ICS)	4.8 minutes	16,692	76,888,623	4.5 hours

Healthcare

Medical devices are becoming increasingly interconnected, which is reflected in the fact that fuzzing protocol tests added during 2016 were related to wireless. Wireless is also the top category in this not-so-very-diverse industry. Wireless and Core IP, for example, make up over three-quarters of the categories in Healthcare. That said, there were a relatively high number of failures against a large number of test runs, suggesting additional investigation is necessary with some of these less mature protocols.

2016 Rank	Protocol suite	Type	Change	2015 Rank
1	TCP for IPv4 Server	Core IP	4	5
2	Traffic Capture Fuzzer	General Purpose	-1	1
3	IPv4	Core IP	-1	2
4	ARP Server	Core IP	-1	3
5	ICMPv4	Core IP	-1	4
6	Traffic Capture Fuzzer with SDK	General Purpose	3	9
7	TCP for IPv4 Client	Core IP	0	7
8	IGMP	Core IP	-2	6
9	A2DP	Wireless	3	12
10	Universal Fuzzer	General Purpose	-2	8
11	RFCOMM	Wireless	2	13
12	DICOM Server	Healthcare	6	18
13	ATT Server	Wireless	-2	11
14	AVRCP	Wireless	new	new
15	ARP Client	Core IP	-12	3
16	Bluetooth LE Advertising Data	Wireless	new	new
17	HFP Unit	Wireless	new	new
18	HFP AG	Wireless	1	19
19	HSP AG	Wireless	1	20
20	SMP	Wireless	new	new

Table 4: Top 20 protocols for Healthcare

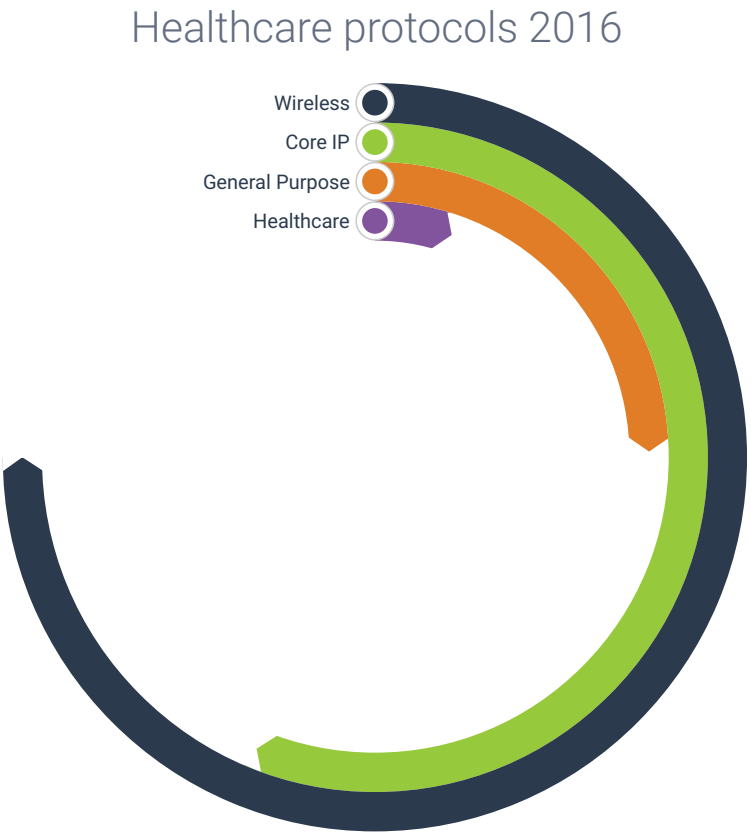

Figure 6: Distribution of protocol categories for Healthcare

Some Healthcare-specific test suite protocols observed in 2016 include these:

- DICOM Server. Digital Imaging and Communications in Medicine (DICOM) is a standard used in medical imaging for handling, storing, printing, and transmitting information. It includes both a file-format definition and a network-communications protocol. There were 1,817 failures recorded in 2016 against 878,364 test runs with an average runtime of 9 minutes.
- AVRCP. Audio/Video Remote Control Profile (AVRCP) is often used in conjunction with Bluetooth technology. There were 12,342 failures against 442,218 test runs in 2016 with an average runtime of 2 hours.
- HFP. This is a Hands-Free Profile (HFP) Bluetooth profile. There were 11,829 failures against 265,283 test runs with an average runtime of 1.2 hours.

The testing times and defects for Healthcare are starting to even out, and the results were fairly consistent, without many outliers.

Protocol suite	Time to first failure	Number of failures	Number of tests	Test runtime (avg)
DICOM Server (Healthcare)	54 seconds	1,817	878,364	9 minutes
AVRCP (Wireless)	10 minutes	12,342	442,218	2 hours
HFP (Wireless)	2.2 minutes	11,829	265,283	1.2 hours

Automotive

The average car today has over 70 electronic control units (ECUs) and over 100 million lines of software code. Adding to the complexity is the increasing connectivity—not just with the internet but with other road vehicles (V2V) as well as stop lights (V2I). Presently, there is significant development to the infotainment unit, the dashboard component that connects to the internet and runs apps to handle a wide variety of internet protocols and input formats. This wide range of input makes the infotainment system a primary target for attackers. Protocols reflecting Core IP and Media make up over 50% of the categories in Automotive.

2016 Rank	Protocol suite	Type	Change	2015 Rank
1	TCP for IPv4 Server	Core IP	4	5
2	Traffic Capture Fuzzer	General Purpose	-1	1
3	IPv4	Core IP	-1	2
4	ARP Server	Core IP	-1	3
5	ICMPv4	Core IP	-1	4
6	Traffic Capture Fuzzer with SDK	General Purpose	4	10
7	MP3	Media	11	18
8	CAN Bus	Bus Technologies	7	15
9	TCP for IPv4 Client	Core IP	-2	7
10	JPEG	Media	new	new
11	802.11 AP	Wireless	5	16
12	IGMP	Core IP	-6	6
13	A2DP	Wireless	6	19
14	PNG	Media	-2	12
15	Universal Fuzzer	General Purpose	-6	9
16	RFCOMM	Wireless	4	20
17	MPEG4	Media	new	new
18	MQTT Client	ICS	-1	17
19	ATT Server	Wireless	-5	14
20	HTTP Client	Core IP	-12	8

Table 5: Top 20 protocols for Automotive

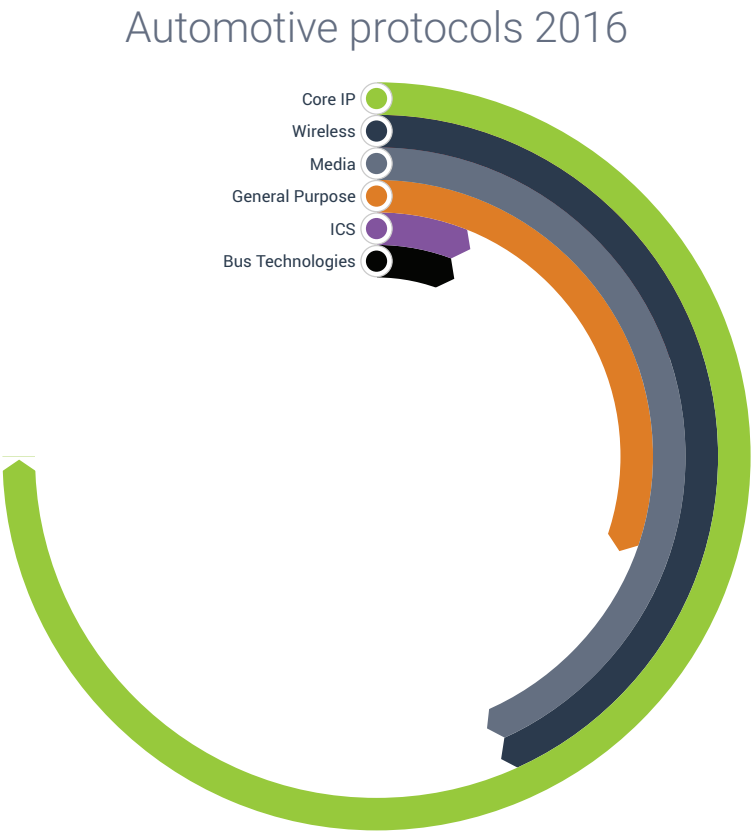

Figure 7: Distribution of protocol categories for Automotive

Some protocols somewhat unique to the automotive industry include these:

- **CAN Bus.** The Controller Area Network (CAN Bus) is defined by the ISO 11898-1 standard. It allows microcontrollers and devices to communicate without a host computer. There were 2,251 failures against 4,472,041 test runs with an average runtime of 15 minutes.
- **A2DP.** The Advanced Audio Distribution Profile (A2DP) focuses on audio streaming. There were 20,255 failures against 2,062,992 test runs with an average runtime of 1.2 hours.
- **RFCOMM.** Radio-frequency communication (RFCOMM) is a simple Bluetooth protocol on top of the L2CAP. There were 5,633 failures against 1,516,260 test runs with an average runtime of 3 hours.

As with Healthcare, the results in Automotive in terms of time to first failure and failures recorded were consistent, with few outliers.

Protocol suite	Time to first failure	Number of failures	Number of tests	Test runtime (avg)
CAN Bus (Bus Technologies)	2.4 minutes	2,251	4,472,041	15 minutes
A2DP (Wireless)	6.4 minutes	20,255	2,062,992	1.2 hours
RFCOMM (Wireless)	25 minutes	5,633	1,516,260	3 hours

Financial Services

Developers of online financial services today practice Agile methodology, with continuous integration and continuous delivery (CI/CD). An attractive feature of Agile is the minimal planning and rapid prototyping that brings new services and features online quickly. The challenge is that Agile methodology relies on automation to speed software delivery, and traditional application security testing tools don't fit well because they aren't automated. Financial Services represents one of the most diverse industries (along with Government), comprising nine protocol categories.

2016 Rank	Protocol suite	Type	Change	2015 Rank
1	TCP for IPv4 Server	Core IP	13	14
2	HTTP Server	Core IP	0	2
3	TCP for IPv4 Server	Core IP	11	14
4	Traffic Capture Fuzzer	General Purpose	-3	1
5	IPv4	Core IP	-2	3
6	ARP Server	Core IP	-2	4
7	DNS Server	Core IP	-2	5
8	ICMPv6	Core IP	0	8
9	ICMPv4	Core IP	2	11
10	IPv6	Core IP	-3	7
11	TLS Server	VPN	-2	9
12	TLS 1.2 Server	VPN	8	20
13	SCTP Server	Cellular Core	6	19
14	TCP for IPv6 Server	Core IP	new	new
15	SMB2 Server	Storage	0	15
16	SNMPv2c	Remote Management	new	new
17	SIP UAS	VoIP	-5	12
18	DHCPv4 Server	Core IP	new	new
19	Ethernet	Metro Ethernet	-13	6
20	CIFS Server	Storage	new	new

Table 6: Top 20 protocols for Financial Services

Figure 8: Distribution of protocol categories for Financial Services

Some protocol tests run within the finance industry are the following:

- SNMPv2c. Simple Network Management Protocol version 2 (SNMPv2) is a protocol for managing computers and devices on an IP network. There were 19,830 failures against 56,619,766 test runs in 2016 with an average runtime of 57 minutes.
- CIFS. Common Internet File System (CIFS) is a protocol to establish an Internet Protocol based file-sharing protocol. There were 2,894 failures against 32,829,319 test runs with an average runtime of 2.2 hours.
- DNS. The Domain Name System (DNS) is a hierarchical, decentralized naming system for computers. It is best known for taking a common name (e.g., Synopsys.com) and matching it with its internet address. There were 13,327 failures against 244,048,739 test runs with an average runtime of 9 hours.

Within Financial Services, the protocols start to show their maturity. DNS, which is used for naming on the internet, was tested 244 million times, resulting in 13,327 failures (less than 1%), with an average of over 9 hours for each test.

Protocol suite	Time to first failure	Number of failures	Number of tests	Test runtime (avg)
SNMPv2c (Remote Management)	14 minutes	19,830	32,829,319	57 minutes
CIFS (Storage)	1 hour	2,894	32,829,319	2.19 hours
DNS (Core IP)	20 minutes	13,327	244,048,739	9 hours

Government

Government is among the most diverse industries (along with Financial Services), with nine protocol categories; however, half are in the Core IP category alone. The maturity of the individual protocols tested in this industry is mixed, with some (e.g., E-LMI) showing no failures and others (e.g., SIP UAS) showing high numbers.

2016 Rank	Protocol suite	Type	Change	2015 Rank
1	HTTP Server	Core IP	1	2
2	TCP for IPv4 Server	Core IP	12	14
3	Traffic Capture Fuzzer	General Purpose	-2	1
4	IPv4	Core IP	-1	3
5	ARP Server	Core IP	-1	4
6	DNS Server	Core IP	-1	5
7	ICMPv6	Core IP	1	8
8	ICMPv4	Core IP	3	11
9	IPv6	Core IP	-1	8
10	TLS Server	VPN	-1	9
11	TLS 1.2 Server	VPN	9	20
12	SCTP Server	Cellular Core	7	19
13	TCP for IPv6 Server	Core IP	new	new
14	SMB2 Server	Storage	1	15
15	SIP UAS	VoIP	-3	12
16	E-LMI	Metro Ethernet	new	new
17	DHCPv4 Server	Core IP	new	new
18	Ethernet	Metro Ethernet	-12	6
19	TLS Client	VPN	-9	10
20	SMB3 Server	Storage	-4	16

Table 7: Top 20 protocols for Government

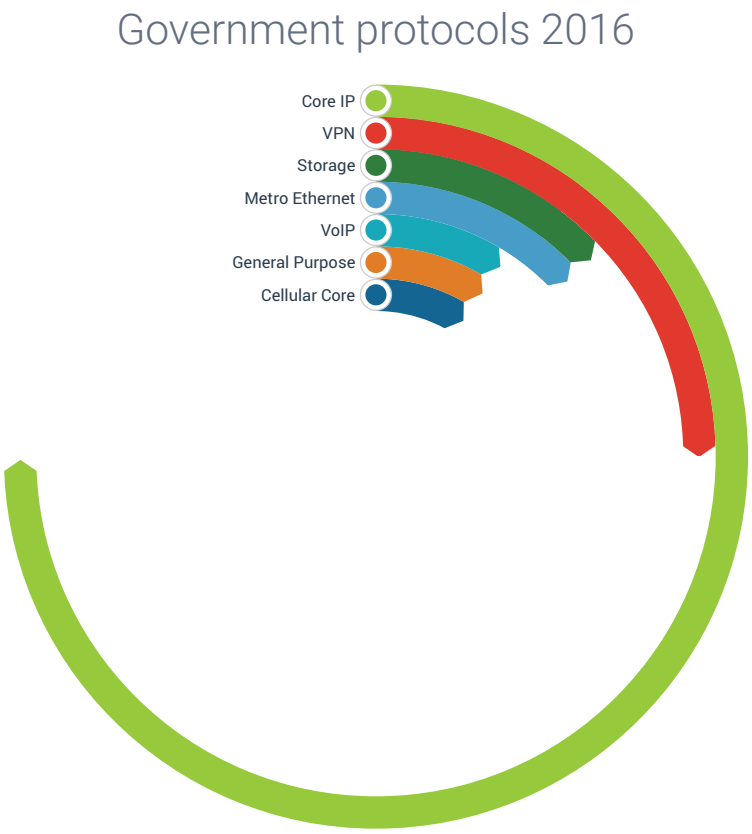

Figure 9: Distribution of protocol categories for Government

Some protocol suites used by government organizations include these:

- **E-LMI.** Ethernet LMI (E-LMI) is an operation, administration, and management (OAM) protocol. It provides signaling standards between routers and switches and maintains information about keepalives, global addressing, IP multicast, and the status of virtual circuits using LMI. There were zero failures against 37,933,443 test runs with an average runtime of 9.3 hours.

- **SCTP.** Stream Control Transmission Protocol (SCTP) is used at the transport layer to transmit several independent streams of chunks in parallel. A common example is transmitting Web page images and Web page text in parallel. There were 26,351 failures in 2016 against 1,516,260 test runs with an average runtime of 5.9 hours.

- **SIP UAS.** The Session Initiation Protocol (SIP) is a communications protocol used for signaling. It sends SIP requests and, as a server, receives other requests and returns responses. There were 320,509 failures last year against 107,431,568 test runs with an average runtime of 3.7 hours.

Like Financial Services, Government includes more mature protocols. The test runs range from 3 to 9 hours, with 0–320,509 failures. The latter number of failures out of 107 million tests is less than 1%.

Protocol suite	Time to first failure	Number of failures	Number of tests	Test runtime (avg)
E-LMI (Metro Ethernet)	0	0	37,933,443	9.3 hours
SCTP (Cellular Core)	1.3 hours	26,351	1,516,260	5.9 hours
SIP UAS (VoIP)	34 minutes	320,509	107,431,568	3.7 hours

The relative maturity of common protocols

The data exemplifies how powerful fuzzing can be at finding potential vulnerabilities. But how long should a fuzz test be run? The response is usually, "It depends."

As shown, certain protocols are more mature, meaning they are implemented better than others. Others are less so, meaning the crashes will start almost from the outset. The State of Fuzzing 2017 identifies a few protocols that can be considered (based on testing data) to be mature, and a few that are virtual greenfields in terms of yielding new vulnerabilities.

Time to first failure (TTFF)

To measure the relative maturity of a given protocol, the time to first failure (TTFF), or the time to the first instance when that a protocol crashes, is used. It is important to note that the crash may or may not be related to a software vulnerability and requires additional investigation.

The true value of TTFF is to identify the average amount of time that is necessary to run a fuzz test before seeing results. In the case of more mature protocols, the length of time is in hours. But with less mature protocols, that time could be as short as a few minutes.

The average time for first failure of all tests in 2016 was about 1 hour (1.4 hours). This represents a very slight decrease over 2015 (1.7 hours), which might be attributable to a more diverse range of tests run in 2016.

5 most mature protocols (average time to first failure, in hours and days)

Protocol suite	2016 time to first failure	2015 time to first failure	2016 test runtime (avg)
TLS Client (Core IP)	9.6 hours	5.4 hours	10.5 hours
SSH2 (Core IP)	6.8 hours	1.9 hours	22.6 hours
SSH Server (Core IP)	4.9 hours	0.7 hours	5.96 hours
802.11 (Core IP)	4.2 hours	1 hour	11.8 hours
ICMPv6 (Core IP)	2.7 hours	1 hour	2.3 days

5 least mature protocols (average time to first failure, in minutes and seconds)

Protocol suite	2016 time to first failure	2015 time to first failure	2016 test runtime (avg)
IEC-61850 MMS (ICS)	6.6 seconds	13.2 seconds	5.7 hours
MODBUS PLC (ICS)	1.8 minutes	6 seconds	35 minutes
SNMP Trap (Remote Management)	1.8 minutes	3.6 minutes	57 minutes
MQTT (ICS)	9.9 minutes	2.1 minutes	1.3 hours
DNP3 (ICS)	14 minutes	3.6 minutes	2.6 hours

The number of test failures

The number of failures is also an interesting metric because it shows where errors are still likely to be found. It is also a misleading metric as the number itself represents not individual vulnerabilities but only the number of crashes recorded.

Without knowing the exact configuration of each system under test (SUT), a single vulnerability may express itself in a crash several times within a test run. Therefore, a manual audit of the crashes is necessary to understand the underlying causes.

Even identifying a vulnerability does not mean that it can be easily exploited. Other circumstances may come into play, or other vulnerabilities might be necessary for an exploit, which is commonly referred to as chaining.

In 2016 we observed

6 protocol tests with 0 failures (this mean that no crashes occurred, not that there were no failures)

Protocol suite	Number of failures	Number of tests	2016 test runtime (avg)
Bluetooth LE Health (Wireless)	0	32,007	5.2 minutes
DHCP v4 Client (Core IP)	0	222,335	13.6 hours
Bluetooth LE (Wireless)	0	334,237	1.6 hours
ARP Client (Core IP)	0	344,433	30 hours
PNG (Media)	0	1,565,835	8.8 hours
E-LMI (Metro Ethernet)	0	37,933,443	9.3 hours

ARP was tested 344,443 times with zero failures recorded for an average runtime of 30 hours per test. This aggressive testing suggests the protocol is implemented correctly and thus unlikely to yield new vulnerabilities. Similarly, we see that E-LMI was tested 37 million times and produced zero failures with an average runtime of 9.3 hours.

A less mature protocol, such as SIP UAS, is complex and difficult to implement correctly. Over 107 million tests were run, with 320,509 failures recorded with an average runtime of 3.7 hours. Another protocol, IPv4, has been around since 1981 and also is hard to implement correctly. There were fewer failures (113,344) against more tests (634,650,464), but the average runtime was low (3.1 hours). This protocol, and others on the list, could all benefit from longer runtimes.

5 protocol tests with the highest number of failures

Protocol suite	Number of failures	Number of tests	2016 test runtime (avg)
IPv4 (Core IP)	113,344	634,650,464	3.1 hours
IPv6 (Core IP)	150,505	128,149,876	8.5 hours
TCP IPv4 Server (Core IP)	173,263	323,100,628	7 hours
HTTP Server (Core IP)	306,571	583,313,532	9.3 hours
SIP UAS (VoIP)	320,509	107,431,568	3.7 hour

Conclusion

The State of Fuzzing 2017 provides quantifiable insight into areas of vulnerability research where further software testing remains necessary. This report qualifies the testing of protocols both by industry and by the relative maturity of those protocols in terms of time to first failure (crash) and number of failures per test run.

The protocols typically associated with ICS showed the most immaturity. Many demonstrated rapid time to first failures, with IEC-61850 MMS measured in a matter of seconds. This has bearing on IoT, as many of the protocols used in ICS are also used in IoT. Clearly, more testing is needed for the protocols within ICS and IoT, as the potential for discovering more vulnerabilities is greater in these industry verticals than in others.

One might think that since Core IP protocols have been around the longest, this group would be the most robust in terms of implementation. While we did observe several Core IP protocols that are well-implemented (e.g., TLS Client, SSH2, and 802.11), many others, such as HTTP Server, a basic protocol for the internet, still generate a significant number of failures, as do TCP for IPv4, IPv4, and IPv6.

Finally, there remain fundamental problems associated with all protocols tested today–namely, not enough testing. For example, one might have a relatively clean implementation of a protocol at the software level and a bad implementation at the firmware level. Therefore, thorough fuzzing of both software and firmware implementations of a protocol is strongly encouraged to ensure overall quality and security.

Best practices to find zero days

Several major organizations have firmly adopted fuzzing as part of their cyber supply chain. If your organization is not currently considering fuzzing, it is probably time to start. Keep in mind the bad guys are already doing it. And as more and more regulations include fuzz testing in their requirements or recommendations, why not be ahead of the curve?

Here are some basic guidelines:

1.Use test suites that include specific areas of input for your software.

- Synopsys has over 250 protocol-based test suites ready to go.

2.Fuzz early.

- Fuzzing occurs in the implementation step of the SDLC and continues through the verification step.

3.Fuzz often.

- Some organizations require a minimum number of faults over a minimum number of hours before shipping a software product. It is also less expensive to remediate vulnerabilities found internally than those reported externally.

Final caveat

Fuzzing won't solve all your vulnerability problems. While it should be used for locating unknown vulnerabilities, it is just one component of what should be a larger vulnerability-management process. In addition to fuzzing, organizations should continue to follow industry best practices, manage known vulnerabilities, and use complementary tools for locating and managing vulnerabilities.

THE SYNOPSYS DIFFERENCE

Synopsys offers the most comprehensive solution for building integrity—security and quality—into your SDLC and supply chain. We've united leading testing technologies, automated analysis, and experts to create a robust portfolio of products and services. This portfolio enables companies to develop customized programs for detecting and remediating defects and vulnerabilities early in the development process, minimizing risk and maximizing productivity. Synopsys, a recognized leader in Application Security Testing, is uniquely positioned to adapt and apply best practices to new technologies and trends such as IoT, DevOps, CI/CD, and the Cloud. We don't stop when the test is over. We offer onboarding and deployment assistance, targeted remediation guidance, and a variety of training solutions that empower you to optimize your investment. Whether you're just starting your journey or well on your way, our platform will help ensure the integrity of the applications that power your business.

For more information, go to www.synopsys.com/software.

SYNOPSYS®

185 Berry Street, Suite 6500
San Francisco, CA 94107 USA

U.S. Sales: 800.873.8193
International Sales: +1 415.321.5237
Email: software-integrity-sales@synopsys.com