FINITE STATE

Understanding the Finite State Risk Profiles

Until now, IoT devices have been like black boxes, meaning that users have no control or visibility into what is running inside them. Finite State's firmware analysis, which illuminates the software and components buried deep inside IoT devices, helps security teams to properly assess the associated risk to their network.

TRANSPARENCY INTO DEVICE RISK

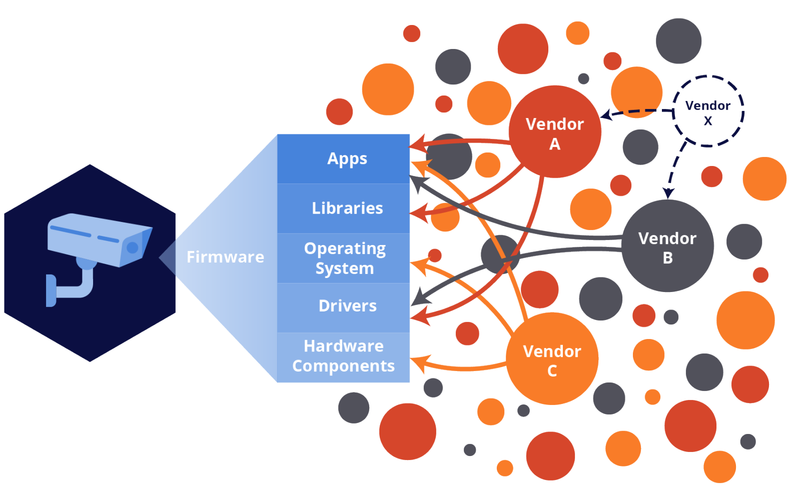
Finite State's risk model is the industry's most comprehensive. Our platform fuses passive network monitoring, firmware analysis, vulnerability data sources, exploit data sources, manufacturer disclosure statements, your own inventory management systems, threat feeds, and more to feed our data models. Further, we continuously update these risk scores and provide historical views, enabling you to understand your current, at the moment posture and how this has changed over time.

Finite State's risk model considers the most dimensions and factors, giving deep insight into the real risk of your deployed device. We can consider the risk based on placement within the network, configuration of the services on the device, the code contained within the firmware, and the type of product to understand all the possible dimensions of risk. We store all of this data and can visualize risk over time as well for each device, allowing you to better understand how your security posture is improving as you respond to our insights.

We use reverse engineering to unpack the firmware image in a device to understand key components, including overall firmware subcomponents and whether the original equipment manufacturer (OEM) follows a secure software development lifecycle.

UNDERSTANDING FIRMWARE CONTENTS

Security teams usually have no idea what is running inside their IoT devices, making it nearly impossible to properly assess risk. To make matters worse, global supply chains and lack of transparency into IoT and other connected devices leaves nearly every organization exposed to potential vulnerabilities buried inside.

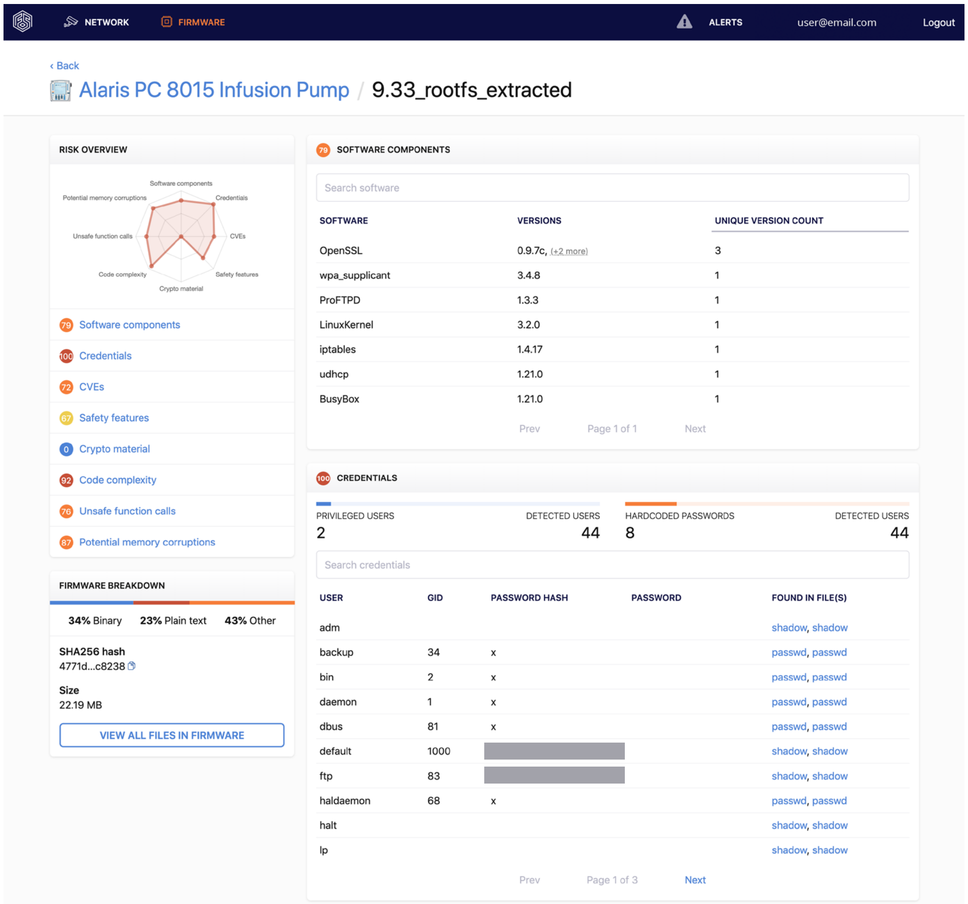


Finite State has built the world's largest firmware reverse engineering system, which has analyzed hundreds of thousands of firmware images (resulting in hundreds of millions of unique files). This approach makes it possible to understand the true risk of devices on any network based on the software that is installed, its configuration, and how the operating system as a whole is configured. To understand the risk any device may pose to the network, we look at key pieces of content in the firmware image, including the software bill of materials and hard-coded credentials and other crypto material that may be present.

FINITE STATE

UNDERSTANDING FIRMWARE CONTENTS

Risk Overview

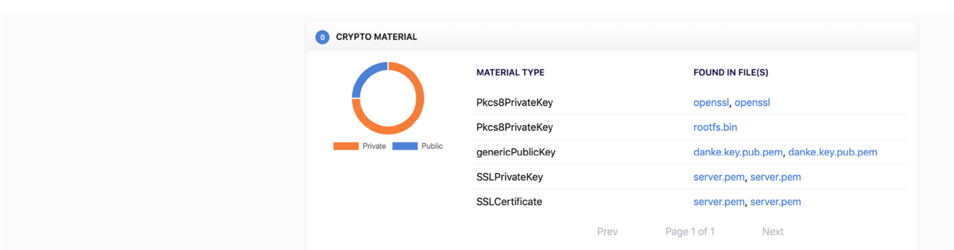


1

Software Bill of Materials

2

Hard-coded Credentials

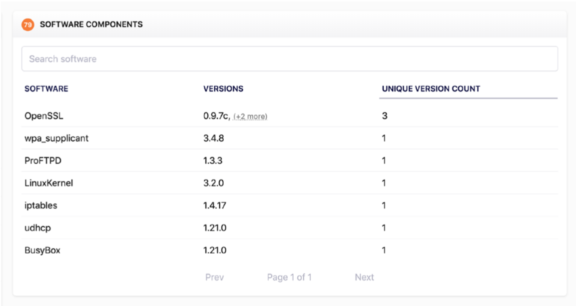


3

Cryptographic Materials

FINITE STATE

1



SOFTWARE COMPONENTS

Search software

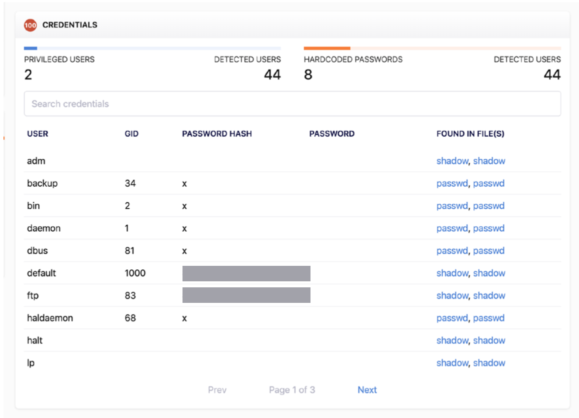
SOFTWARE	VERSIONS	UNIQUE VERSION COUNT
OpenSSL	0.9.7c, (+2 more)	3
wpa_supplicant	3.4.8	1
ProFTPD	1.3.3	1
LinuxKernel	3.2.0	1
iptables	1.4.17	1
udhcp	1.21.0	1
BusyBox	1.21.0	1

Prev Page 1 of 1 Next

SOFTWARE BILL OF MATERIALS

Finite State unpacks the software bill of materials, providing visibility into what's running on an IoT device, including binaries like Bash, BusyBox, Curl, dropbear, and even OpenSSL. Not only does this help inform the risk profile, but understanding the software bill of materials allows us to more positively identify products and software running on the network based on firmware-version unique characteristics.

2



CREDENTIALS

PRIVILEGED USERS 2 DETECTED USERS 44 HARDCODED PASSWORDS 8 DETECTED USERS 44

Search credentials

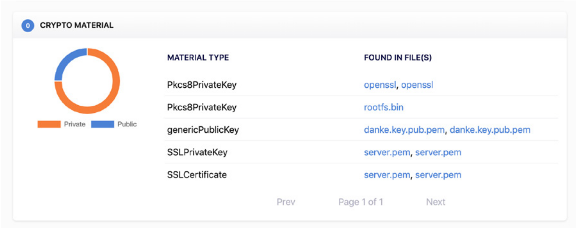
USER	GID	PASSWORD HASH	PASSWORD	FOUND IN FILE(S)
adm				shadow, shadow
backup	34	x		passwd, passwd
bin	2	x		passwd, passwd
daemon	1	x		passwd, passwd
dbus	81	x		passwd, passwd
default	1000	[redacted]		shadow, shadow
ftp	83	[redacted]		shadow, shadow
haldaemon	68	x		passwd, passwd
halt				shadow, shadow
lp				shadow, shadow

Prev Page 1 of 3 Next

HARD-CODED CREDENTIALS

Automated analysis capabilities locate, extract, and attempt to recover plaintext credentials for all accounts on the system. Having a full accounting of the credentials in a firmware often leads to the discovery of potential backdoors that increase the risk to the network.

3



CRYPTO MATERIAL

MATERIAL TYPE	FOUND IN FILE(S)
Pkcs8PrivateKey	openssl, openssl
Pkcs8PrivateKey	rootfs.bin
genericPublicKey	danke.key.pub.pem, danke.key.pub.pem
SSLPrivateKey	server.pem, server.pem
SSLCertificate	server.pem, server.pem

Prev Page 1 of 1 Next

CRYPTOGRAPHIC MATERIALS

Similar to hard-coded credentials, cryptographic material contained in a firmware image is highly problematic. The presence of materials such as private keys and authorized key files can produce backdoors allowing unintended access to the device. The presence of poorly configured cryptographic settings like the presence of standardized host key files may weaken the security envelope of devices, as these should be unique per device, not common across firmware.

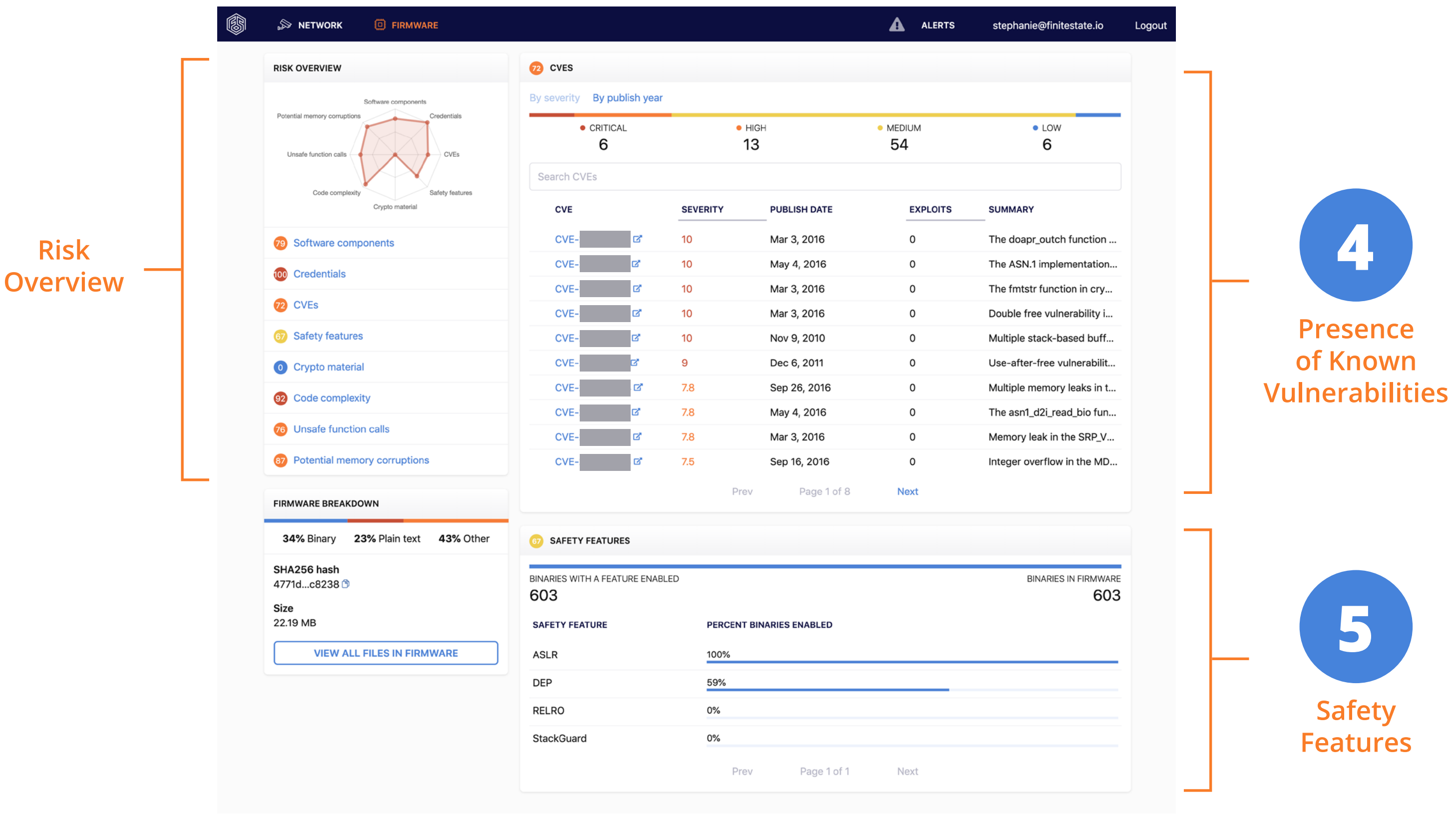
Worse yet, because of the supply chain for these devices, numerous devices are marketed and sold from completely different companies and contain these same cryptographic materials inside.

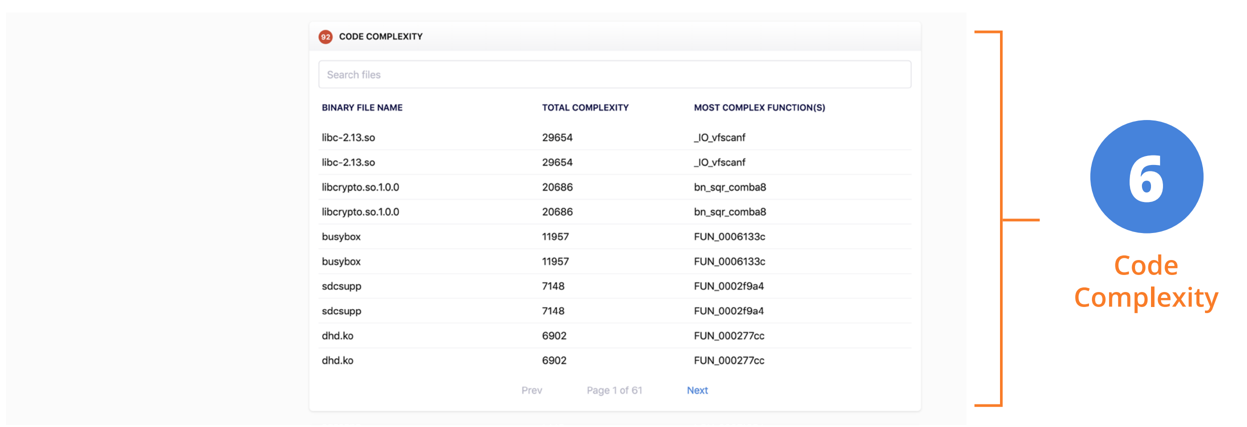
Note that seeing this in the firmware does not necessarily mean there was malicious intent. In most cases crypto materials are included as part of the debugging process. Unfortunately, these materials can be forgotten.

FINITE STATE

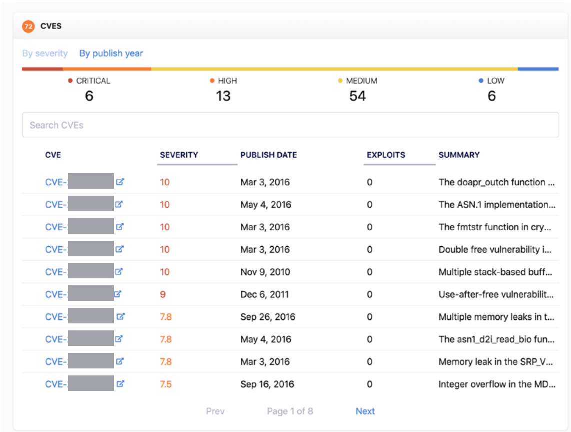
GAUGING QUALITY AND SAFETY

Firmware analysis is also critical to understanding a product manufacturer's secure software development lifecycle. Finite State analyzes key factors that indicate the relative security of any software development lifecycle by observing typical indications of secure development, including how the presence of known vulnerabilities, whether third-party software is used, the use of binary safety features, code complexity, the number of safe and unsafe function calls, and memory corruptions.





4



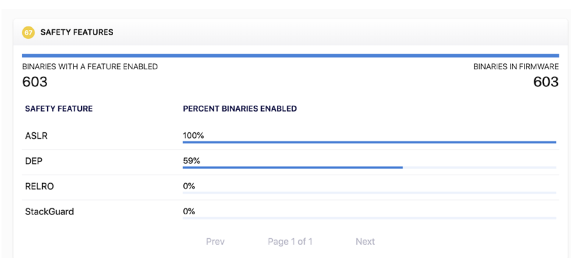
PRESENCE OF KNOWN VULNERABILITIES

Based on the Software Bill of Materials that includes all known third-party binaries, the operating system, and awareness of the product itself, Finite State identifies all known vulnerabilities in this software automatically. Data from these vulnerability data sources is automatically deduplicated and presented to users. Finite State also correlates information from the vulnerability database about the risk of the vulnerability with known exploit data, allowing users to understand how these vulnerabilities are being used by real-world malicious actors.

This equips you with the same level of visibility used by an attacker who is aware of all software on the device and capable of using these known vulnerabilities to link together an attack chain. One vulnerability can be used for access to the device, one can be used for privilege escalation, and so forth until the device is fully compromised.

This level of visibility into known vulnerabilities baked into a device is only possible through firmware analysis. If you look up this specific device within the National Vulnerability Database, you'll notice that is has no CVEs associated with it. But if you look INSIDE the device using the Finite State Platform, you can see that there are more than a thousand CVEs associated with the packages that are present.

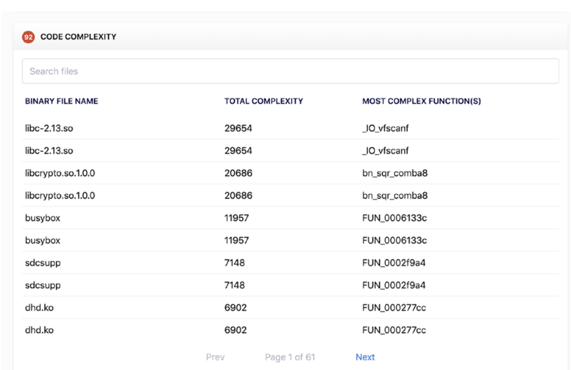
5



SAFETY FEATURES

Another component that we look at in our analysis are the binary safety features. A binary is generally compiled from source code into machine executable code. Most modern compilers will come with safety features to prevent address lookups, buffer overflows, and things of that nature. These features in modern compilers are turned on by default—so when we see that these are not enabled on binaries, we can assume that someone has actively turned these features off. That may have been done maliciously, or it may have been done to make the existing code work. We cannot determine intent; however, we can see if these compiler level protections are turned on consistently to protect against malicious attacks.

6

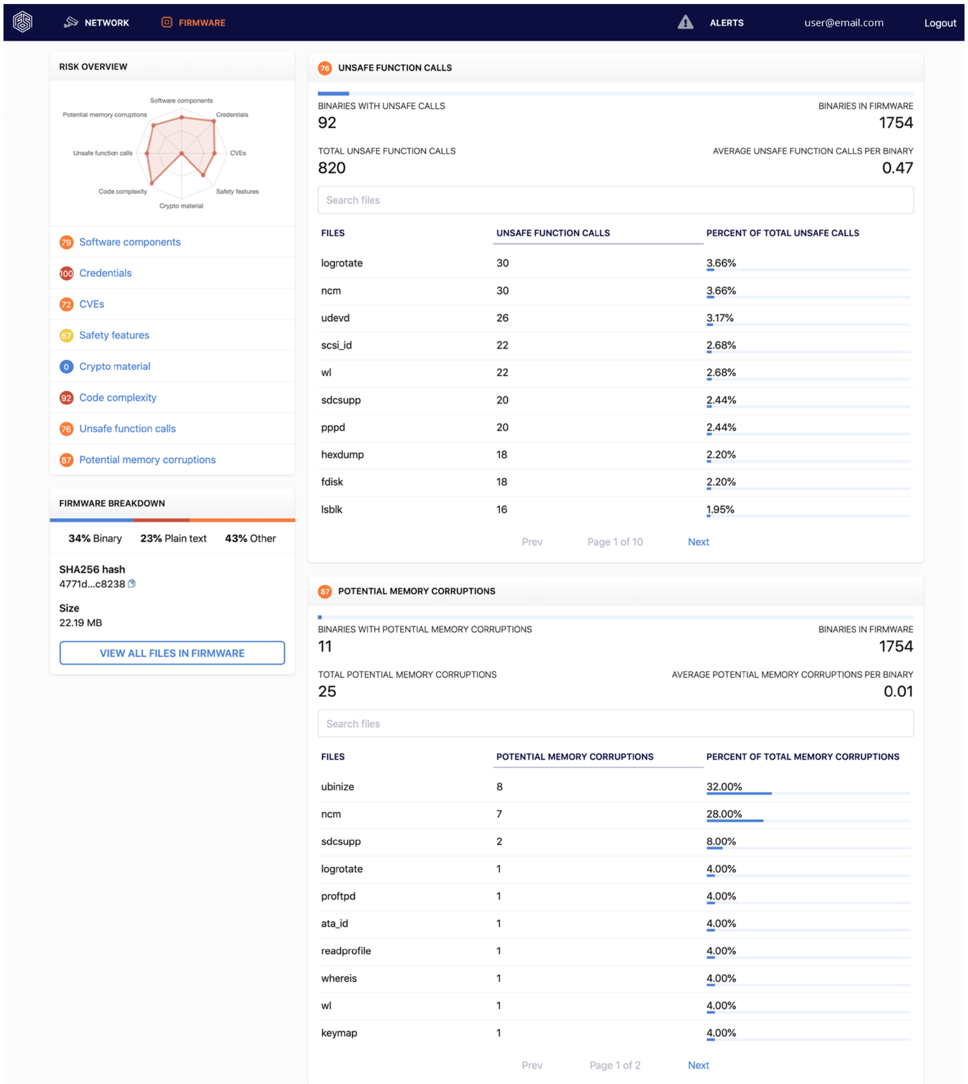


CODE COMPLEXITY

Code complexity can help analysts understand the risk profile and stability estimations of any unit of code. This particular metric effectively looks at the number of different decisions that can be made in a unit of code. When this score is higher, there are more logical paths to follow, which means there is a higher level of difficulty to adequately test the software. Software that is more difficult to test has been shown in many studies to have a higher risk of defects, which correlates with security vulnerabilities. Simply put, simpler code is more secure.

GAUGING QUALITY AND SAFETY

Risk Overview



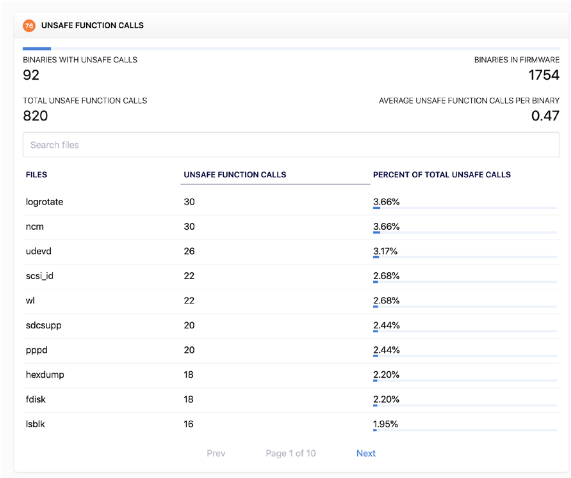
7

Unsafe Function Calls

8

Memory Corruption

7

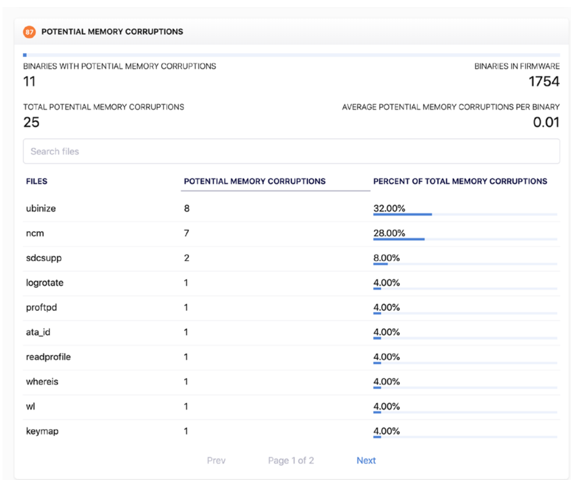


FILES	UNSAFE FUNCTION CALLS	PERCENT OF TOTAL UNSAFE CALLS
logrotate	30	3.66%
ncm	30	3.66%
udevd	26	3.17%
scsi_id	22	2.68%
wl	22	2.68%
sdcsupp	20	2.44%
pppd	20	2.44%
hexdump	18	2.20%
fdisk	18	2.20%
lsblk	16	1.95%

UNSAFE FUNCTION CALLS

In programming languages like C, there are a series of legacy functions like strcpy that are considered unsafe and have modern analogs like strncpy. These legacy functions have long been known to be insecure for many years. The secure functions, in many cases, have been available for more than a decade. Finite State identifies the first- and third-party binaries being used and whether they use these functions. If manufacturers include numerous unsafe function calls in their own code, that suggests they are struggling to build secure software. The presence of unsafe function calls in third-party code is also likely a string indicator of how thoroughly they vet and maintain third-party tools. Together, this metric provides a better understanding of the priority level given to security throughout the software development lifecycle.

8



FILES	POTENTIAL MEMORY CORRUPTIONS	PERCENT OF TOTAL MEMORY CORRUPTIONS
ubinize	8	32.00%
ncm	7	28.00%
sdcsupp	2	8.00%
logrotate	1	4.00%
proftpd	1	4.00%
ata_id	1	4.00%
readprofile	1	4.00%
whereis	1	4.00%
wl	1	4.00%
keymap	1	4.00%

MEMORY CORRUPTION

A memory corruption is a type of vulnerability that may occur when memory is altered without an explicit assignment, meaning that the items stored at that memory location can be modified. Finite State has developed analysis tools to automatically find previously unknown 0-day memory corruption vulnerabilities, allowing us to understand how well the software development team implemented memory management practices, as well as what unknown memory corruption vulnerabilities live within the software.

TRANSPARENCY IMPROVES SECURITY

With years of offensive cyber operations experience, our team understands that attackers know more about your devices than you do. They gain this knowledge by looking inside the firmware, and they find trivially exploitable vulnerabilities. The Finite State Platform has been designed from the ground up to enable you to look deep inside the devices on your network, gain an in-depth understanding of the risks buried inside the firmware, and establish a new era of transparency. Transparency undeniably improves security, and by leveraging our unique capabilities in firmware reverse engineering, comprehensive risk modeling, and advanced detection models, you can stay ahead of attackers for the first time.